AppleTalk Remote Access Modem Toolkit Version 1.0

R0129LL/B

Apple Computer, Inc.
20525 Mariani Avenue, M/S 33-G
Cupertino, CA 95014
(408) 996-1010
TLX 171-576

To reorder products, please call:
1-800-282-2732 (in the United States)
1-800-637-0029 (in Canada)
1-408-562-3910 (International)

AppleTalk Remote Access Modem Scripting Guide

Apple Computer, Inc.

20525 Mariani Avenue
Cupertino, CA 95014-6299
(408) 996-1010

Simultaneously published in the United States and Canada.

Contents

Preface

This guide describes how to write a modem script for use with AppleTalk Remote Access, an application program that allows you to use a Macintosh computer to access another Macintosh or AppleTalk network services over standard telephone lines. A *modem script* is a set of instructions that tells your computer how to interact with a modem so that calls can be initiated and received.

What you need to know to use this guide

To get the most out of this guide, you should know the basics of how to use a Macintosh computer. You should also have a good understanding of telecommunications and modem operation. Although you don't need an in-depth knowledge of computer programming, some knowledge of basic programming concepts would be helpful.

How to use this guide

Here's how to use this guide successfully to write a modem script:

- Read Chapter 1 to find out how to use the Modem Workshop tool to create and test your script.
- Read the section "Script Conventions" in Chapter 2 to find out the basic elements and structure of a Communication Control Language (CCL) file. CCL is the language you use to write your script.
- Read the section "Functions Your Script Must Perform" in Chapter 2 to find out the way your modem must work to establish an AppleTalk Remote Access connection.
- In the section "A Sample Modem Script" in Chapter 2, look at the sample script and examine how each section of code works. Once you understand how the sample script functions, you can construct your own script.

- Refer to Chapter 3 for information about the the syntax and use of the commands you need to write your script. The commands are listed in alphabetical order for easy reference.
- Look up error code information in Chapter 4.

What you need to get started

To write your script, your Macintosh computer must be running system software version 7.0 or later. You must have HyperCard® 2.0 or later and AppleTalk Remote Access software installed on your computer. You also need the Modem Workshop tool, which is on the *Modem ToolKit* disk that accompanies this guide.

You may want to refer to the *AppleTalk Remote Access User's Guide* and the documentation that came with your modem for more information.

Modem scripts already available

A number of modem scripts have already been written for use with AppleTalk Remote Access and are available automatically when you install the Remote Access software. If you have a modem from the following list, you don't need to write a script.

Once you install the Remote Access software, these scripts reside in your Extensions folder (which is located in the System Folder). You may want to look at some of the scripts to see how to write various sections of your own script. Also, you may be able to use an existing script as a template.

Scripts for the following modems are included with AppleTalk Remote Access (see your *AppleTalk Remote Access User's Guide* for further details):

- Abaton InterFax 24/96
- Apple Data Modem (supports all 2400-bps Apple modems)
- DSI 9624 LE/LE Plus
- Farallon Remote V.32
- Global Village TelePort
- Hayes Smartmodem 2400
- Hayes ULTRA 96
- Microcom MacModem V.32
- Microcom MicroPort 1042
- MultiTech MultiModemV32
- Practical Peripherals 2400SA

- Practical Peripherals 9600SA
- Prometheus 2400M
- Prometheus ProModem Ultima
- Supra SupraModem 2400
- Telebit T1600
- US Robotics Courier 2400e
- US Robotics Courier V.32bis

1 Using Modem Workshop

The Modem Workshop tool is a HyperCard® stack that enables you to write and test your modem script. The Workshop includes an editor that allows you to create and modify your script, and the Workshop's testing environment lets you view what is going in and out of the Macintosh computer's serial port (that is, the commands sent to the modem and the result codes returned by the modem).

If your script has an error in it when you attempt to use it, the AppleTalk Remote Access program will not tell you what specifically is wrong with the script. Therefore, it is essential that you test your script with Modem Workshop so you can accurately locate and correct bugs.

This chapter describes how to use the Modem Workshop tool to create and test your modem script.

Installing the Modem Workshop tool

Before you install and use the Modem Workshop tool, make sure you've installed the AppleTalk Remote Access program.

To install the Modem Workshop tool, follow these steps:

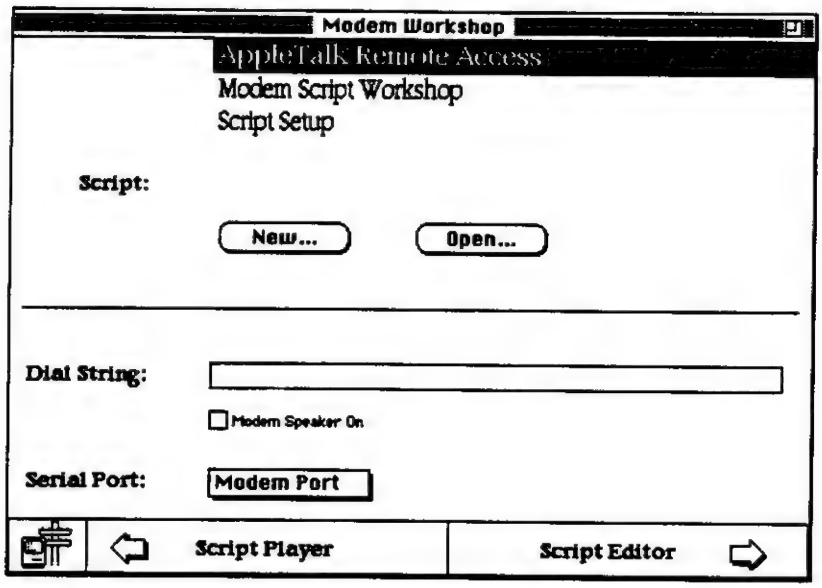
1. **Insert the disk called *Modem ToolKit* into your floppy disk drive and open it.**
2. **Open the Modem Workshop folder and drag the Modem Workshop HyperCard stack to your hard disk.**
3. **Open the folder called Put Into Extensions Folder and drag the CCL Tool to your Extensions folder (which resides in the System Folder).**
4. **Close the Put Into Extensions Folder and then the Modem Workshop folder, and drag the *Modem ToolKit* disk to the Trash to eject it.**

Creating a new script

To create a new script, follow these steps:

1. **Double-click the Modem Workshop icon on your hard disk.**

The Script Setup card appears.

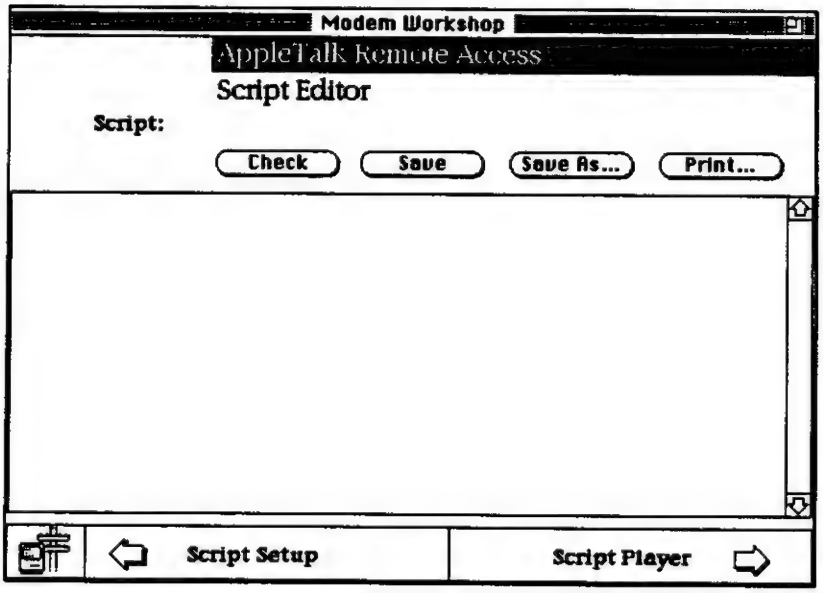


2 **Click the New button.**

The New button creates an empty script in the Modem Workshop editor.

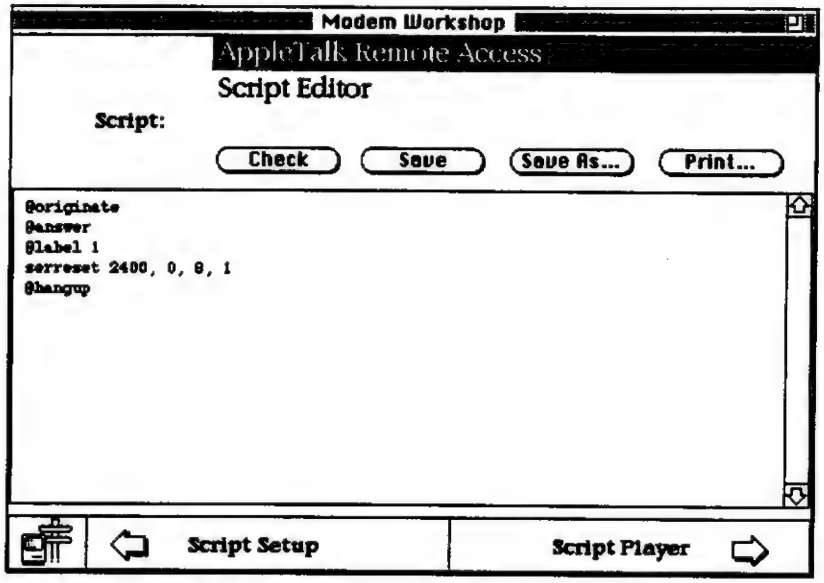
3 **Click the Script Editor button in the lower-right corner of the Script Setup card.**

The Script Editor card appears.



4 **Enter your script text in the space provided in the card.**

Chapter 2, "Writing the Script," provides step-by-step details on writing your script.



5 **If you want to check the syntax, click the Check button.**

Syntax information appears in a message window beneath the Script Editor card, as in the following example:

6 **Click the Save As button to save your script text.**

A dialog box appears that allows you to give your script a name. Type a name for the document and then click the Save button.

7 **If you want a copy of your script text, click the Print button.**

8 **To test your script, follow the instructions in the section "Testing a Script" (later in this chapter). Or, if you want to quit the Modem Workshop tool, select Quit HyperCard from the File menu.**

Modifying a script

To modify an already existing script, follow these steps:

1 **Double-click the Modem Workshop icon on your hard disk.**

The Script Setup card appears.

2 **Click the Open button.**

A dialog box appears with a directory of the current folder or disk.

3 **Double-click the name of the script file you want to edit.**

The filename you select appears next to "Script:" in the Script Setup card, as in the following example:

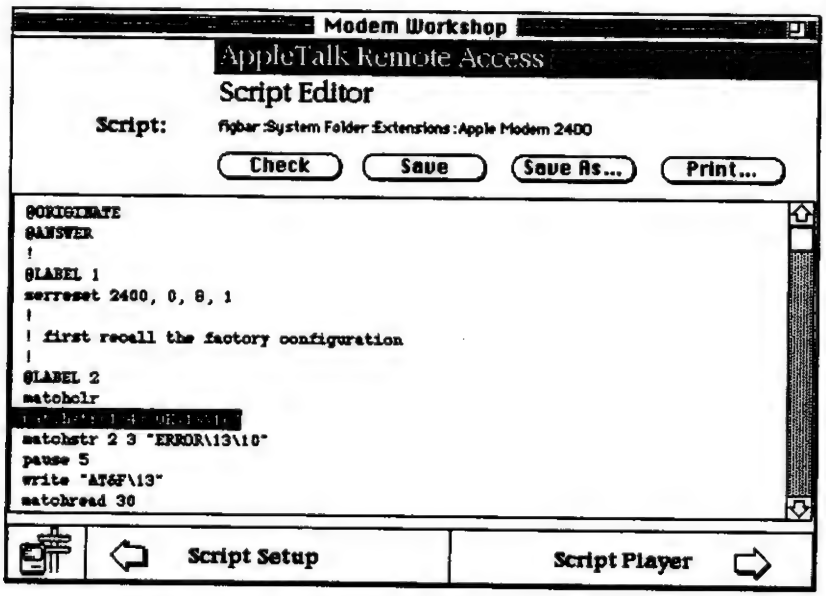
Script: apollo:System Folder:Extensions:Apple Modem 2400

4 **Click the Script Editor button in the lower-right corner of the Script Setup card.**

The Script Editor card appears.

5 **Edit the script text.**

Use standard Macintosh editing techniques to edit the script.



6 **If you want to check the syntax, click the Check button.**

Syntax information appears in a message window beneath the Script Editor window, as in the following example:

7 **Click the Save button to save your script text.**

8 **To quit the Modem Workshop tool, select Quit HyperCard from the File menu.**

Testing a script

To test a script, follow these steps. Note that to test a script thoroughly, you should call a modem that you know works correctly.

1 **Double-click the Modem Workshop icon on your hard disk.**

The Script Setup card appears.

2 **Enter the phone number of the modem you want to call in the Dial String text box.**

All modem dialing string modifiers can be used in the string (for instance, `P` for pulse dialing). See your modem documentation for more information about dialing string modifiers.

3 **Click the Modem Speaker On checkbox if you want to hear auditory feedback from the modem.**

This feature allows you to test the modem with the modem speaker on or off. An X in the box indicates the speaker is on.

4 **Use the Serial Port pop-up menu to select the port to which your modem is connected.**

5 **Click the Open button.**

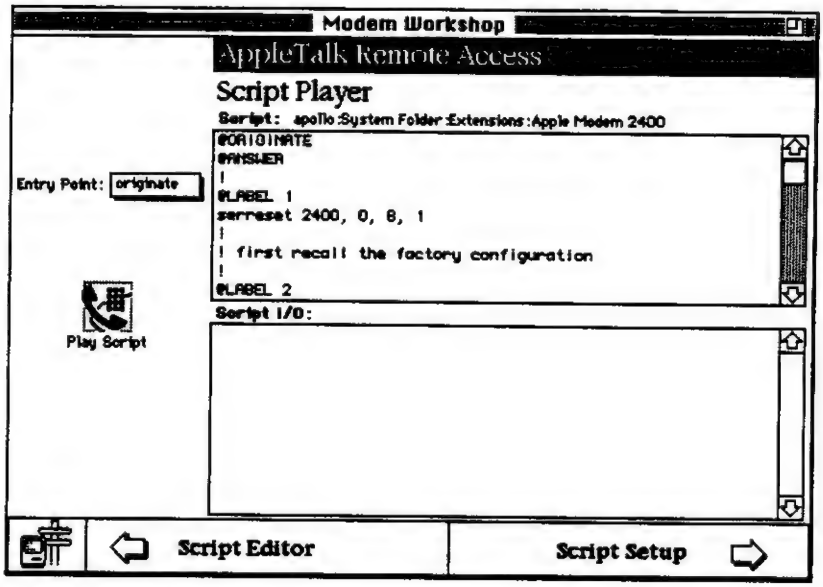
A dialog box appears with a directory of the current folder or disk.

6 **Double-click the name of the script file you want to edit.**

The filename you select appears next to "Script:" in the Script Setup card.

7 **Click the Script Player button in the lower-left corner of the Script Setup card.**

The Script Player card appears.

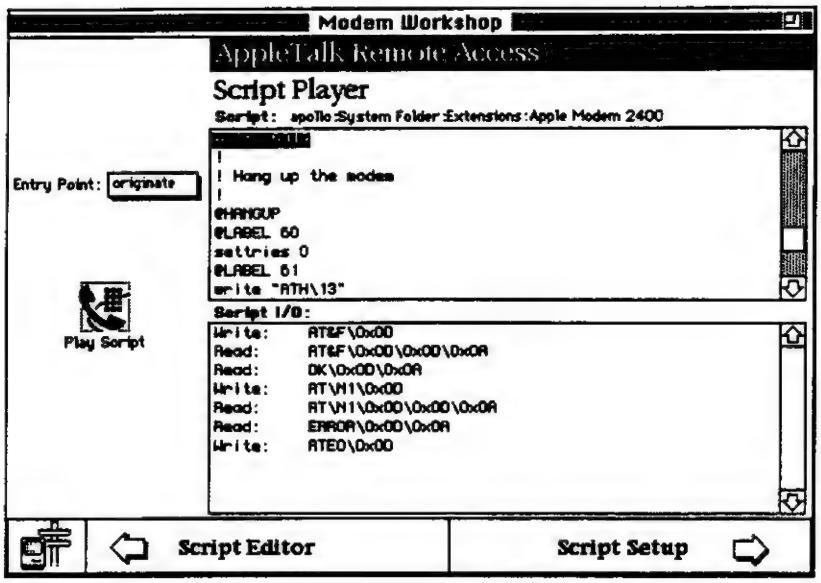


8 **Press the Entry Point pop-up menu and choose the mode that you want the script to play in.**

You can play the script in originate, answer, and hangup mode. Note that when you play the script in originate or answer mode, you must then play it in hangup mode (the tool will not automatically hang up for you).

9 **Click the Play Script icon.**

The tool plays the script and displays the commands sent to the modem and the result codes returned by the modem in the Script I/O window, as shown in the following figure.



Errors that occur are displayed in a message window beneath the Script Player card, as in the following example:

10 **Click the Script Editor button in the lower-left corner of the Script Player card and edit your script as needed.**

See the section "Modifying a Script" earlier in this chapter for details.

11 **To quit the Modem Workshop tool, select Quit HyperCard from the File menu.**

Using your script with AppleTalk Remote Access

After testing and debugging your script, you need to make it available for use with the AppleTalk Remote Access program:

- **Place your script in your Extensions folder (which resides in the System Folder).**

  You're now ready to use your script with AppleTalk Remote Access. Follow the instructions in the *AppleTalk Remote Access User's Guide.*

2 Writing the Script

This chapter explains the conventions your AppleTalk Remote Access script must follow and describes the basic functions it must perform. Also included in this chapter is a sample modem script and a detailed explanation of how it works. You can use this script as a model when writing your own script.

Script conventions

You write modem scripts using the Communication Control Language (CCL), a programming language that configures and controls your modem. The following sections describe the basic elements and structure of a CCL file.

Instructions

Each line of CCL code consists of one instruction that is made up of a command and its parameter(s) (if any). A parameter is the part of an instruction that specifies how the instruction is carried out. Modem commands are used as parameters of CCL commands. For example, in the command

```
write AT&F\13
```

`write` is a CCL command and `AT&F` is a modem command. This command tells the CCL interpreter to send the modem command `AT&F` followed by a carriage return (ASCII code 13) to the modem.

The CCL interpreter reads your script from left to right and from top to bottom. It reads straight through, from beginning to end, unless you tell it otherwise.

Comments

You can insert explanatory comments into your script. To enter a comment, start the line with an exclamation point (!). You may also want to use a comment line to make your script more readable; to do so, type an exclamation point with no text and press Return.

Capitalization

The CCL interpreter does not care how instructions are capitalized. Therefore,

```
@LABEL
@Label
@label
@laBel
```

are all valid instructions.

Labels

Labels are used to mark locations in the script. Other script commands, such as `JUMP`, transfer control to locations in the script marked by the `@LABEL` command. For instance, `JUMP 13` tells the CCL interpreter to jump to label 13 and start executing the commands after the `@LABEL 13` command.

String formats

A string refers to any group of characters. This section describes the format of strings used in CCL commands.

To delimit a string, you can use single quotes ('), double quotes ("), or the default delimiters. The default delimiter characters are space, return, tab, comma, and semicolon. If any carriage returns, default delimiter characters, or nonprinting characters are to be included in the string, you need to delimit the string with quotes.

CCL strings may include references to varStrings, which are strings passed in as parameters to the script. The varStrings used by AppleTalk Remote Access are:

`varString1`	the dial string
`varString2`	the modem speaker flag

CCL strings may include nonprinting characters such as null, tab, and return. To specify these characters in a string, you must use an escape character. The CCL interpreter recognizes two escape characters, the backslash (\) and the caret (^). The backslash is used to include characters by specifying the decimal representation of the ASCII character (decimal numbers 00 to 99 are valid). The backslash is also used to include the backslash and caret characters in strings. The caret is used to insert a variable string or an ASK string into another string. Here are some examples of how the backslash and caret are used:

`\13`	inserts a carriage return (0x0D) into the string
`\00`	inserts the null character (0x00) into the string
`\\`	inserts the \ character (0x53) into the string
`\^`	inserts the ^ character into the string
`^1`	inserts variable string 1 into the dial string
`^*`	inserts the ASK string into the string

Here are some string examples:

`'this is a test'`	yields	this is a test
`thisisatest`	yields	thisisatest
`"this is a test"`	yields	this is a test
`this is a test`	yields	this
`"this \34\73\83\34 a test"`	yields	this "IS" a test

Functions your script must perform

Your script must tell the modem how to initiate, answer, and terminate a call. You must enter one of the following commands in your script to mark where it should start playing when it has to perform one of these functions:

- `@ORIGINATE` (initiate a call)
- `@ANSWER` (answer a call)
- `@HANGUP` (terminate a call)

The following sections describe the tasks you must make the modem perform in each of these modes of operation. (Note that in the sections "Initiating a Call (@ORIGINATE)" and "Answering a Call (@ANSWER)" steps 1 and 2 are the same.)

Initiating a call (@ORIGINATE)

When you signal the AppleTalk Remote Access program to initiate a call, it plays the script at the `@ORIGINATE` entry point. The following steps describe how to write your script to make the modem initiate a call.

1 **Configure the Macintosh computer's serial port.**

Use the `SERRESET` command to reset the serial port's communication parameters. This command passes certain parameters to the serial driver, such as speed of the connection and the number of bits to be used for stop, start, and parity.

2 Configure the modem.

Configure the modem options as follows:

a. Recall the factory default configuration settings specified by the manufacturer.

b. Configure the modem to ignore Data Terminal Ready (DTR). In some cases, such as when you need to reset a modem that has stopped responding to commands, you use DTR to reset the modem. DTR is a control signal that tells the modem whether the Macintosh is ready to communicate with it. If the modem is configured to use DTR, you need the proper cable.

c. Put the modem in direct-connect mode, which disables speed conversion. Direct-connect mode is a mode of data transfer in which the Macintosh-to-modem interface speed must match the modem-to-modem speed. When in direct-connect mode, the modem adjusts its serial interface speed to match its modem interface speed.

d. Turn off the error-control mechanism. AppleTalk Remote Access has error control built into it; if you do not turn off error control, AppleTalk Remote Access will not function properly. An error-correction protocol is a set of agreed-upon rules used to check for and correct errors in data transmission over a connection between modems. The most widely used of these protocols is the Microcom Networking Protocol (MNP).

◆ **Note** If your modem supports the Trellis error protocol, which is part of the V.32 standard, it should be left on. ◆

e. Turn off flow control. Flow control is a modem feature that regulates the rate at which data is sent and received.
   - Disable software flow control (XON/XOFF).
   - Hardware flow control (Request To Send/Clear To Send) is disabled when you put the modem in direct-connect mode. Note that hardware flow control may be necessary for some modems; for example, if you cannot disable speed conversion by putting the modem in direct-connect mode. For hardware flow control, you need the proper cable and you must turn on flow control in the Macintosh using the `HSRESET` command.

f. Turn local echo off. When local echo is on, the modem sends commands it receives back to the Macintosh.

g. Turn the modem speaker on or off according to the value specified in the Remote Access Setup control panel.

3 **Dial the phone number and wait for the result.**

Display the dialed phone number in the Remote Access Status window and the Activity Log. If the call is successfully made, the modem returns a result indicating that the connection is established and gives the speed of the connection (for instance, `CONNECT 9600`).

4 **If the call fails, return a result code indicating what happened.**

For example, the result code `-6022` should be returned when the line is busy.

5 **Indicate that a connection has been established.**

Display a message such as "Communicating at 2400 bps" in the Remote Access Status window.

6 **Exit the script so that the AppleTalk Remote Acess program can continue with its internal execution.**

Answering a call (@ANSWER)

When the AppleTalk Remote Access program decides that it's time to answer a call, it plays the script at the `@ANSWER` entry point. The following steps describe how to write your script to answer a call.

1 **Configure the Macintosh computer's serial port.**

Use the `SERRESET` command to reset the serial port's communication parameters. This command passes certain parameters to the serial driver, such as speed of the connection and the number of bits to be used for stop, start, and parity.

2 **Configure the modem.**

Configure the modem options as follows:

a. Recall the factory default configuration settings specified by the manufacturer.

b. Configure the modem to ignore Data Terminal Ready (DTR). In some cases, such as when you need to reset a modem that has stopped responding to commands, you use DTR to reset the modem. DTR is a control signal that tells the modem whether the Macintosh is ready to communicate with it. If the modem is configured to use DTR, you need the proper cable.

c. Put the modem in direct-connect mode, which disables speed conversion. Direct-connect mode is a mode of data transfer in which the Macintosh-to-modem interface speed must match the modem-to-modem speed. When in direct-connect mode, the modem adjusts its serial interface speed to match its modem interface speed.

d. Turn off the error-control mechanism. AppleTalk Remote Access has error control built into it; if you do not turn off error control, AppleTalk Remote Access will not function properly. An error-correction protocol is a set of agreed-upon rules used to check for and correct errors in data transmission over a connection between modems. The most widely used of these protocols is the Microcom Networking Protocol (MNP).

◆ **Note** If your modem supports the Trellis error protocol, which is part of the V.32 standard, it should be left on. ◆

e. Turn off flow control. Flow control is a modem feature that regulates the rate at which data is sent and received.
   - Disable software flow control (XON/XOFF).
   - Hardware flow control (Request To Send/Clear To Send) is disabled when you put the modem in direct-connect mode. Note that hardware flow control may be necessary for some modems; for example, if you cannot disable speed conversion by putting the modem in direct-connect mode. For hardware flow control, you need the proper cable and you must turn on flow control in the Macintosh using the `HSRESET` command.

f. Turn local echo off. When local echo is on, the modem sends commands it receives back to the Macintosh.

g. Turn the modem speaker on or off according to the value specified in the Remote Access Setup control panel.

3 Enable auto-answering and wait for the result.

If the call is successfully made, the modem returns a result indicating that the connection is established and gives the speed of the connection (for instance, `CONNECT 9600`).

4 If the call fails, return a result code indicating what happened.

For example, the result code `-6021` should be returned when the modem cannot establish a carrier.

5 **Detect an incoming call.**

On an incoming call, a `RING` result code is issued by the modem. At this time, it is important for the script to run the `USERHOOK 1` command. This command informs AppleTalk Remote Access that the serial port is busy answering a call, which prevents the serial port from being given up to another application. The `NOTE` command should be used to inform the user of an incoming call.

6 **Indicate that a connection has been established.**

Display a message such as "Communicating at 2400 bps" in the Remote Access Status window.

7 **Exit the script so that the AppleTalk Remote Acess program can continue with its internal execution.**

Terminating a call (@HANGUP)

To terminate a call, the AppleTalk Remote Access program plays the script at the `@HANGUP` entry point. The hangup part of the script is played to terminate a connection whenever the `@ORIGINATE` or `@ANSWER` parts of the script have been played, regardless of the result of the script. The hangup part of the script is also played when another application gains control of the serial driver or the system is shut down and AppleTalk Remote Access terminates answer mode.

◆ **Note** AppleTalk Remote Access has a callback feature (see the *AppleTalk Remote Access User's Guide* for more information). For callback to work in all cases, the script must enable the modem to hang up within 15 seconds. This is the minimum time that the answering side will wait before issuing the callback. It is possible that the answering side will take longer to issue the callback. The maximum time the calling side will wait for a callback is 80 seconds. ◆

The following steps describe how to make the modem perform the appropriate tasks when the script is in hangup mode:

1 **If hardware handshaking is used in the `@ORIGINATE` or `@ANSWER` script, turn off hardware handshaking.**

Handshaking is an exchange of signals that is used to coordinate data transfer between two devices. Use the `HSRESET` command to turn off hardware handshaking.

2 **Put the modem in command mode if there's no response to the** `HANGUP` **command.**

3 **Recall the factory default configuration settings.**

This ensures that the modem responds correctly the next time the script is played.

4 **Turn off auto-answering.**

The modem will not answer the phone until call answering is enabled.

A sample modem script

This section presents the modem script for the Apple Data Modem 2400. The text to the left of the script indicates a corresponding section following the script that explains in detail that part of the code.

Mark originate and answer mode

```
@ORIGINATE
@ANSWER
!
```

Configure the serial port

```
@LABEL 1
serreset 2400, 0, 8, 1
!
! first recall the factory configuration
!
```

Recall the factory settings

```
@LABEL 2
matchclr
matchstr 1 4 "OK\13\10"
matchstr 2 3 "ERROR\13\10"
pause 5
write "AT&F\13"
matchread 30
jump 59
!
! The &F command failed, must be original!
portable modem, try Z
!
```

(continued)➡

Recall factory settings for Macintosh Portable Data Modem 2400

```
@LABEL 3
matchclr
matchstr 1 6 "OK\13\10"
write "ATZ\13"
matchread 30
jump 59
!
! Set up the configuration: direct
! connection mode, no mnp, no speed
! conversion
!
```

Set up the configuration

```
@LABEL 4
matchstr 1 5 "OK\13\10"
matchstr 2 6 "ERROR\13\10"
write "AT\\N1\13"
matchread 30
!
! Next, turn off flow control
!
```

Turn off flow control

```
@LABEL 5
matchclr
matchstr 1 6 "OK\13\10"
write "AT&K0\13"
matchread 30
jump 59
!
! Turn echo off
!
```

Turn echo off

```
@LABEL 6
matchclr
matchstr 1 7 "OK\13\10"
write "ATE0\13"
matchread 30
jump 59
!
! If speaker on flag is true, jump to
! label 8; else turn off the speaker
!
```

Turn the modem's speaker on or off

```
@LABEL 7
ifstr 2 8 "1"
matchstr 1 8 "OK"
write "ATM0\13"
matchread 30
jump 59
!
! The modem is ready so enable answering,
! or originate a call
!
```

Enable answering or originate a call

```
@LABEL 8
pause 5
ifANSWER 30
note "Dialing ^1" 3
write "ATDT^1\13"
!
```

Jump according to result codes

```
@LABEL 9
matchstr 1 11 "CONNECT 1200"
matchstr 2 12 "CONNECT 2400"
matchstr 3 50 "NO CARRIER"
matchstr 4 50 "ERROR"
matchstr 5 52 "NO DIALTONE"
matchstr 6 53 "BUSY"
matchstr 7 54 "NO ANSWER"
matchread 700
jump 59
!
```

Indicate that a connection was made

```
@LABEL 11
note "Communicating at 1200 bps." 2
serreset 1200, 0, 8, 1
jump 15
!
@LABEL 12
note "Communicating at 2400 bps." 2
serreset 2400, 0, 8, 1
!
```

Transfer control

```
@LABEL 15
ifANSWER 16
pause 30
```

Exit

```
@LABEL 16
exit 0
!
! @ANSWER
! Set up the modem to answer
```

Set up the modem to answer

```
@LABEL 30
write "ATS0=1\13"
matchstr 1 31 "OK\13\10"
matchread 30
jump 59
!
```

Jump according to result codes

```
@LABEL 31
matchstr 1  32 "RING"
matchstr 2  11 "CONNECT 1200"
matchstr 3  12 "CONNECT 2400"
matchstr 4  50 "NO CARRIER"
matchstr 5  50 "ERROR"
matchstr 6  52 "NO DIALTONE"
matchstr 7  53 "BUSY"
matchstr 8  54 "NO ANSWER"
matchread 700
jump 31
!
```

(continued)➨

Ensure that no call is attempted

```
@LABEL 32
userhook 1
note "Answering phone…" 2
jump 31
!
! 50: error messages
! No carrier
```

Exit and return an error

```
@LABEL 50
exit -6021
! No Dial Tone
@LABEL 52
exit -6020
! Busy
@LABEL 53
exit -6022
! No Answer
@LABEL 54
exit -6023
! Modem error
@LABEL 59
exit -6019
!
! Hang up the modem
!
```

Mark hangup mode

```
@HANGUP
```

Initialize the tryCounter variable

```
@LABEL 60
settries 0
```

Issue the hangup command

```
@LABEL 61
write "ATH\13"
matchclr
matchstr 1 62 "NO CARRIER\13\10"
matchstr 2 62 "OK\13\10"
matchstr 3 62 "ERROR\13\10"
matchread 30
inctries
iftries 3 62
! no response, try escape sequence
write "+++"
matchclr
matchstr 1 61 "OK\13\10"
matchread 15
jump 61
!
! Recall the factory settings
!
```

Recall the factory settings

```
@LABEL 62
matchclr
matchstr 1 64 "OK\13\10"
matchstr 2 63 "ERROR\13\10"
write "AT&F\13"
matchread 30
!
! the &F failed, must be original
! portable modem, try Z!
```

Recall factory settings for Macintosh Portable Data Modem 2400

```
@LABEL 63
matchclr
matchstr 1 65 "OK\13\10"
write "ATZ\13"
matchread 30
!
! Turn off auto answer
!
```

Turn off auto-answering

```
@LABEL 64
write "ATS0=0\13"
matchclr
matchstr 1 65 "OK\13\10"
matchread 30
!
```

Exit

```
@LABEL 65
exit 0
```

Mark originate and answer mode

```
@ORIGINATE
@ANSWER
```

This part of the script indicates where the script should begin playing when it is originating or answering a call.

Configure the serial port

```
@LABEL 1
serreset 2400, 0, 8, 1
```

This part of the script configures the serial port. The `serreset` command sets the baud rate to 2400, parity to 0 (which turns it off), data bits to 8, and stop bits to 1. Parity, data bits, and stop bits are set to the values indicated across all modems. The value for baud rate can vary (300, 1200, 2400, 4800, 9600, 19,200, or 57,600). The Apple Data Modem transmits at 1200 and 2400 bps.

Recall the factory settings (originate mode)

```
@LABEL 2
matchclr
matchstr 1 4 "OK\13\10"
matchstr 2 3 "ERROR\13\10"
pause 5
write "AT&F\13"
matchread 30
jump 59
```

This part of the script recalls the modem's factory default configuration.

- The `matchclr` command erases all match strings from the CCL interpreter's match string list.
- Each `matchstr` command places a match string in the match string list, along with an identifying number and a label where control should jump if the string is read from the modem. In this instance, if the modem responds with the OK status string `OK\13\10`, execution jumps to label 4.
- The `pause` command tells the modem to suspend execution for 0.5 seconds (the time parameter with the `pause` command is measured in tenths of a second).
- The `write` command recalls the factory configuration settings from the modem (check your modem documentation to make sure this `AT&F` is the appropriate command). The modem should issue a string in response indicating that it has understood and performed the command.
- The `matchread` command reads the response from the modem and compares it to the match strings placed in the match string list by the `matchstr` commands. If the response from the modem is the `OK\13\10`, execution jumps to label 4 to configure other options on the modem. If the response is `ERROR\13\10`, execution jumps to label 3 to make another attempt to recall the factory configuration. The time parameter with the `matchread` command is measured in tenths of a second (therefore, this `matchread` searches for a match within 3.0 seconds).
- If there is no response from the modem, the `jump` command tells the CCL interpreter to skip directly to label 59, which terminates the execution of the script and returns error code `-6019` (the modem is not responding).

Recall factory settings for Macintosh Portable Data Modem 2400 (originate mode)

```
@LABEL 3
matchclr
matchstr 1 6 "OK\13\10"
write "ATZ\13"
matchread 30
jump 59
```

The Macintosh Portable Data Modem 2400 does not support the `AT&F` command (described in the previous section). If you have this modem, the `AT&F` command will fail to recall factory configuration settings and control will jump to label 3 to try again with the `ATZ` command.

- The `matchclr` command erases all match strings from the CCL interpreter's match string list.
- The `matchstr` command places the OK status string `OK\13\10` in the match string list and specifies that control should jump to label 6 if this string is read from the modem.
- The `write` command recalls the factory configuration settings from the modem (the `ATZ` command works with the Macintosh Portable Data Modem 2400). The modem should issue a string in response indicating that it has understood and performed the command.
- The `matchread` command reads the response from the modem and compares it to the match strings placed in the match string list by the `matchstr` command. If the response from the modem is `OK\13\10`, execution jumps to label 6 to configure other options on the modem. The time parameter with the `matchread` command is measured in tenths of a second (therefore, this `matchread` searches for a match within 3.0 seconds).
- If there is no response from the modem, the `jump` command tells the CCL interpreter to skip directly to label 59, which terminates the execution of the script and returns error code `-6019` (the modem is not responding).

Set up the configuration

```
@LABEL 4
matchstr 1 5 "OK\13\10"
matchstr 2 6 "ERROR\13\10"
write "AT\\N1\13"
matchread 30
jump 59
```

This part of the script sets up the modem configuration.

- Each `matchstr` command places a match string in the match string list, along with an identifying number and a label where control should jump if the string is read from the modem. In this instance, if the modem responds with the OK status string `OK\13\10`, execution jumps to label 5.
- The `write` command tells the modem to connect in direct mode with MNP and speed conversion off. On several types of modems (for example, the Apple Data Modem and the Microcom MacModem V.32), you can accomplish this by using the `AT\\N1` command. Other modems may require more than one command (for example, the Hayes ULTRA 96 and the Telebit T1600) or no command at all (for example, the Global Village TelePort). The modem should issue a string in response indicating that it has understood and performed the command.
- The `matchread` command reads the response from the modem and compares it to the match strings placed in the match string list by the `matchstr` command. If the response from the modem is `OK\13\10`, execution jumps to label 5. If the response is `ERROR\13\10`, execution jumps to label 6. The time parameter with the `matchread` command is measured in tenths of a second (therefore, this `matchread` searches for a match within 3.0 seconds).
- If there is no response from the modem, the `jump` command tells the CCL interpreter to skip directly to label 59, which terminates the execution of the script and returns error code `-6019` (the modem is not responding).

Turn off flow control

```
@LABEL 5
matchclr
matchstr 1 6 "OK\13\10"
write "AT&K0\13"
matchread 30
jump 59
```

This part of the script turns off flow control.

- The `matchclr` command erases all match strings from the CCL interpreter's match string list.
- The `matchstr` command places the OK status string `OK\13\10` in the match string list and specifies that control should jump to label 6 if this string is read from the modem.
- The `write` command disables local flow control (check your modem documentation to make sure that `AT&K0` is the appropriate command). The modem should issue a string in response indicating that it has understood and performed the command.
- The `matchread` command reads the response from the modem and compares it to the match strings placed in the match string list by the `matchstr` commands. If the response from the modem is `OK\13\10`, execution jumps to label 6. The time parameter with the `matchread` command is measured in tenths of a second (therefore, this `matchread` searches for a match within 3.0 seconds).
- If there is no response from the modem, the `jump` command tells the CCL interpreter to skip directly to label 59, which terminates the execution of the script and returns error code `-6019` (the modem is not responding).

Turn echo off

```
@LABEL 6
matchclr
matchstr 1 7 "OK\13\10"
write "ATE0\13"
matchread 30
jump 59
```

This part of the script turns local echo off.

- The `matchclr` command erases all match strings from the CCL interpreter's match string list.

- The `matchstr` command places the OK status string `OK\13\10` in the match string list and specifies that control should jump to label 7 if this string is read from the modem.
- The `write` command turns off local echo (check your modem documentation to make sure that `ATE0` is the appropriate command). The modem should issue a string in response indicating that it has understood and performed the command.
- The `matchread` command reads the response from the modem and compares it to the match strings placed in the match string list by the `matchstr` commands. If the response from the modem is `OK\13\10`, execution jumps to label 7. The time parameter with the `matchread` command is measured in tenths of a second (therefore, this `matchread` searches for a match within 3.0 seconds).
- If there is no response from the modem, the `jump` command tells the CCL interpreter to skip directly to label 59, which terminates the execution of the script and returns error code `-6019` (the modem is not responding).

Turn the modem's speaker on or off

```
@LABEL 7
ifstr 2 8 "1"
matchstr 1 8 "OK"
write "ATM0\13"
matchread 30
jump 59
```

This part of the script turns the modem's speaker on or off. When the Remote Access Setup control panel is configured (see the *AppleTalk Remote Access User's Guide* for details), there is an option to turn the modem speaker on or off. If the On option is selected, `"1"` is passed into the script as string variable 2. If the Off option is selected, `"0"` is passed into the script as string variable 2.

- The `ifstr` command compares two strings. Here, it compares the contents of string variable 2 with `"1"`. If they match (the speaker is on), control jumps to label 8. If they do not match (the speaker is off), execution continues sequentially and the command to turn the speaker off is issued with the `write` command.
- The `matchstr` command places `OK` in the match string list and specifies that control should jump to label 8 if this string is read from the modem.
- The `write` command turns the modem speaker off (check your modem documentation to make sure that `ATM0` is the appropriate command). The modem should issue a string in response indicating that it has understood and performed the command.

- The `matchread` command reads the response from the modem and compares it to the match strings placed in the match string list by the `matchstr` command. If the response from the modem is OK, execution jumps to label 8. The time parameter with the `matchread` command is measured in tenths of a second (therefore, this `matchread` searches for a match within 3.0 seconds).
- If there is no response from the modem, the `jump` command tells the CCL interpreter to skip directly to label 59, which terminates the execution of the script and returns error code `-6019` (the modem is not responding).

Enable answering or originate a call

```
@LABEL 8
pause 5
ifANSWER 30
note "Dialing ^1" 3
write "ATDT^1\13"
jump 59
```

Once the modem options have been configured, the script can enable the modem to answer or to initiate a call. If the script is playing in answer mode, a command sends the CCL interpreter to another location in the script. If the script is playing in originate mode, the script informs the Remote Access user that a call is being placed and instructs the modem to dial the appropriate number.

- The `pause` command tells the modem to suspend execution for 0.5 seconds (the time parameter with the `pause` command is measured in tenths of a second).
- The `ifANSWER` command causes the script to continue execution at label 30 if the script is playing in answer mode.
- The `note` command displays the message "Dialing [number]" (the number entered in the Remote Access connection document is passed in for `^1`). Message level 3 sends the message to both the Activity Log and the Remote Access Status window. For more information about the Remote Access connection document, Activity Log, and Status window, see the *AppleTalk Remote Access User's Guide.*
- The `write` command tells the modem to tone-dial the number passed into parameter `^1` (check your modem documentation to make sure that `ATDT` is the appropriate command). Note that tone dialing (as opposed to pulse dialing) is appropriate for U.S. phones but may not be appropriate in other countries.
- If there is no response from the modem, the `jump` command tells the CCL interpreter to skip directly to label 59, which terminates the execution of the script and returns error code `-6019` (the modem is not responding).

Jump according to result codes (originate mode)

```
@LABEL 9
matchstr 1 11 "CONNECT 1200"
matchstr 2 12 "CONNECT 2400"
matchstr 3 50 "NO CARRIER"
matchstr 4 50 "ERROR"
matchstr 5 52 "NO DIALTONE"
matchstr 6 53 "BUSY"
matchstr 7 54 "NO ANSWER"
matchread 700
jump 59
```

This part of the script tells the CCL interpreter, when it is running in originate mode, what label to go to in the script based on the result codes the modem returns after a call has been dialed.

- Each `matchstr` command places a match string in the match string list, along with an identifying number and a label where control should jump if the string is read from the modem. For instance, if the modem responds with `CONNECT 1200`, execution jumps to label 11.
- The `matchread` command reads the response from the modem and compares it to the match strings placed in the match string list by the `matchstr` commands. If the response from the modem matches a string in the match string list, execution jumps to the label of the matching match string. The time parameter with the `matchread` command is measured in tenths of a second (therefore, this `matchread` searches for a match within 70.0 seconds).
- If there is no response from the modem, the `jump` command tells the CCL interpreter to skip directly to label 59, which terminates the execution of the script and returns error code `-6019` (the modem is not responding).

Indicate that a connection was made

```
@LABEL 11
note "Communicating at 1200 bps." 2
serreset 1200, 0, 8, 1
jump 15
!
@LABEL 12
note "Communicating at 2400 bps." 2
serreset 2400, 0, 8, 1
```

This part of the script performs actions associated with two result codes, specified in the previous section of code, that indicate a connection has been established. For each result code (for example, `CONNECT 1200`), the script displays a message in the Remote Access Status window informing the user that a connection has been made, resets the serial driver, and transfers control to the part of the script that in turn transfers control back to the Remote Access program.

- The `note` command in label 11 displays the message "Communicating at 1200 bps" and the `note` command in label 12 displays the message "Communicating at 2400 bps." Message level 2 sends the message to the Remote Access Status window only.
- The `serreset` command in label 11 sets the baud rate to 1200, parity to 0, data bits to 8, and stop bits to 1. The `serreset` command in label 12 sets the baud rate to 2400, parity to 0, data bits to 8, and stop bits to 1.
- If there is no response from the modem, the `jump` command tells the CCL interpreter to skip directly to label 15, which transfers control to the Remote Access program.

Transfer control

```
@LABEL 15
ifANSWER 16
pause 30
```

If the script is playing in originate or answer mode it now must transfer control back to the Remote Access program. If the script is playing in answer mode, control skips to label 16. If the script is playing in originate mode, it waits 3.0 seconds before going on to label 16, which allows the answering side of the connection to set up before the originating side transfers or requests data.

- The `ifANSWER` command causes the script to continue execution at label 16 if the script is playing in answer mode.
- The `pause` command tells the modem to suspend execution for 3.0 seconds (the time parameter with the `pause` command is measured in tenths of a second).

Exit

```
LABEL 16
exit 0
```

This command causes the CCL interpreter to quit the script. The Remote Access program continues with its internal execution. An exit code of 0 means that no error has occurred and the script has finished execution without error. Any nonzero error code means that something has gone wrong with the connection.

Set up the modem to answer

```
@LABEL 30
write "ATS0=1\13"
matchstr 1 31 "OK\13\10"
matchread 30
jump 59
```

This part of the script executes when the script plays in answer mode. (In label 8, described in the section "Enable Answering or Originate a Call," the `ifANSWER` command causes the script to continue execution at this label if the script is playing in answer mode.) The script sets up the modem to answer the phone line by setting the value of S register 0 to 1, which tells the modem to answer the phone after one ring. S registers are registers in the modem's memory (the amount of which is modem-specific) that modify certain modem characteristics such as timing parameters. The S commands are used to assign values to various registers in the modem's memory.

- The `write` command tells the modem to answer the phone on the first ring (check your modem documentation to make sure that `ATS=1` is the appropriate command). The modem should issue a string in response indicating that it has understood and performed the command.
- The `matchstr` command places the OK status string `OK\13\10` in the match string list and specifies that control should jump to label 31 if this string is read from the modem.

- The `matchread` command reads the response from the modem and compares it to the match strings placed in the match string list by the `matchstr` command. If the response from the modem matches a string in the match string list, execution jumps to the label of the matching match string. The time parameter with the `matchread` command is measured in tenths of a second (therefore, this `matchread` searches for a match within 3.0 seconds).
- If there is no response from the modem, the `jump` command tells the CCL interpreter to skip directly to label 59, which terminates the execution of the script and returns error code `-6019` (the modem is not responding).

Jump according to result codes (answer mode)

```
@LABEL 31
matchstr 1 32 "RING"
matchstr 2 11 "CONNECT 1200"
matchstr 3 12 "CONNECT 2400"
matchstr 4 50 "NO CARRIER"
matchstr 5 50 "ERROR"
matchstr 6 52 "NO DIALTONE"
matchstr 7 53 "BUSY"
matchstr 8 54 "NO ANSWER"
matchread 700
jump 31
```

This part of the script tells the CCL interpreter what label to go to based on the result codes the modem returns after a call has been dialed.

- Each `matchstr` command places a match string in the match string list, along with an identifying number and a label where control should jump if the string is read from the modem. For instance, if the modem responds with `RING`, execution jumps to label 32.
- The `matchread` command reads the response from the modem and compares it to the match strings placed in the match string list by the `matchstr` commands. If the response from the modem matches a string in the match string list, execution jumps to the label of the matching match string. The time parameter with the `matchread` command is measured in tenths of a second (therefore, this `matchread` searches for a match within 70.0 seconds).
- If there is no response from the modem, the `jump` command tells the CCL interpreter to skip to the beginning of this section of code (label 31).

Ensure that no call is attempted

```
@LABEL 32
userhook 1
note "Answering phone..." 2
jump 31
```

This part of the script executes when the modem returns result code `RING`, which occurs when the phone rings. This code ensures that an outgoing call will not be attempted while the modem is answering a call.

- The `userhook` command marks the connection as active when a ring is indicated by the modem (parameter `1` is passed to the user hook).
- The `note` command displays the message "Answering phone...." Message level 2 sends the message to the Remote Access Status window only.
- The `jump` command tells the CCL interpreter to go back to label 31, which waits for the next result code from the modem.

Exit and return an error

```
@LABEL 50
exit -6021
@LABEL 52
exit -6020
@LABEL 53
exit -6022
@LABEL 54
exit -6023
@LABEL 59
exit -6019
```

Commands throughout the script transfer control to these error-handling routines if something goes wrong. The `exit` commands terminate execution of the script and return one of the following result codes:

- `exit -6021` returns the error result code for no carrier.
- `exit -6020` returns the error result code for no dial tone.
- `exit -6022` returns the error result code for a busy signal.
- `exit -6023` returns the error result code for no answer.
- `exit -6019` returns the error result code for a modem error.

Mark hangup mode

```
@HANGUP
```

This part of the script indicates where the script should begin playing when it is hanging up a call, both on the initiating and receiving sides of the connection. Note that this part of the script will vary between modems. Your script should

- instruct the modem to hang up the phone
- reconfigure the modem so that the script may be played over from the beginning

Initialize the tryCounter variable

```
@LABEL 60
settries 0
```

This part of the script initializes the tryCounter variable to 0. The `settries` command works in conjuntion with the `inctries` and `iftries` commands specified in the next section of code.

Issue the hangup command

```
@LABEL 61
write "ATH\13"
matchclr
matchstr 1 62 "NO CARRIER\13\10"
matchstr 2 62 "OK\13\10"
matchstr 3 62 "ERROR\13\10"
matchread 30
inctries
iftries 3 62
write "+++"
matchclr
matchstr 1 61 "OK\13\10"
matchread 15
jump 61
```

This part of the script issues the hangup command to the modem.

- The `write` command issues the hangup command (check your modem documentation to make sure that `ATH` is the appropriate command). The modem should issue a string in response indicating that it has understood and performed the command.

- The `matchclr` command erases all match strings from the CCL interpreter's match string list.
- Each `matchstr` command places a match string in the match string list, along with an identifying number and a label where control should jump if the string is read from the modem. If the modem responds with `NO CARRIER`, `OK`, or `ERROR`, execution jumps to label 62.
- The `matchread` command reads the response from the modem and compares it to the match strings placed in the match string list by the `matchstr` commands. If the response from the modem matches a string in the match string list, execution jumps to the label of the matching match string. The time parameter with the `matchread` command is measured in tenths of a second (therefore, this `matchread` searches for a match within 3.0 seconds).
- The `inctries` command increases the tryCounter variable by 1 if none of the result codes (`NO CARRIER`, `OK`, or `ERROR`) is returned.
- The `iftries` command compares the parameter `3` with the tryCounter variable. If tryCounter is greater than or equal to 3, the script continues execution at label 62. This command causes the loop to iterate three times.
- If the modem is not responding to the hangup command, the `write` command issues the escape sequence (+++) to try and put the modem into command mode (check your modem documentation to make sure that `+++` is the appropriate command). The modem should issue a string in response indicating that it has understood and performed the command. After the escape sequence is issued, some modems may not be ready to accept another command. If this is the case, you may need to insert a `pause` command into the code.
- The `matchclr` command erases all match strings placed in the match string list by the previous `matchstr` command.
- The `matchstr` command places `OK\13\10` in the match string list and specifies that control should jump to label 61 if this string is read from the modem. Therefore, if the modem recognizes and responds to the escape sequence, control shifts back to the top of the loop (label 61).
- The `matchread` command reads the response from the modem and compares it to the match strings placed in the match string list by the `matchstr` commands. If the response from the modem is `OK\13\1`, execution jumps to label 61. The time parameter with the `matchread` command is measured in tenths of a second (therefore, this `matchread` searches for a match within 1.5 seconds).
- If there is no response from the modem, the `jump` command tells the CCL interpreter to skip directly to label 61 to restart the loop.

Recall the factory settings (hangup mode)

```
@LABEL 62
matchclr
matchstr 1 64 "OK\13\10"
matchstr 2 63 "ERROR\13\10"
write "AT&F\13"
matchread 30
```

When the script exits after hangup, the modem should be reconfigured so that it responds correctly the next time the script is played. This part of the script recalls the modem's factory default configuration.

- The `matchclr` command erases all match strings from the CCL interpreter's match string list.
- Each `matchstr` command places a match string in the match string list, along with an identifying number and a label where control should jump if the string is read from the modem. In this instance, if the modem responds with the OK status string `OK\13\10`, execution jumps to label 64.
- The `write` command recalls the factory configuration settings from the modem (check your documentation to make sure that `AT&F` is the appropriate command). The modem should issue a string in response indicating that it has understood and performed the command.
- The `matchread` command reads the response from the modem and compares it to the match strings placed in the match string list by the `matchstr` commands. If the response from the modem is `OK\13\10`, execution jumps to label 64. If the response is `ERROR\13\10`, execution jumps to label 63 to make another attempt to recall the factory configuration. The time parameter with the `matchread` command is measured in tenths of a second (therefore, this `matchread` searches for a match within 3.0 seconds).

Recall factory settings for Macintosh Portable Data Modem 2400 (hangup mode)

```
@LABEL 63
matchclr
matchstr 1 65 "OK\13\10"
write "ATZ\13"
matchread 30
```

The Macintosh Portable Data Modem 2400 does not support the `AT&F` command (described in the previous section). If you have this modem, the `AT&F` command will fail to recall factory configuration settings and control will jump to label 63 to try again with the `ATZ` command.

- The `matchclr` command erases all match strings from the CCL interpreter's match string list.
- The `matchstr` command places the OK status string `OK\13\10` in the match string list and specifies that control should jump to label 65 if this string is read from the modem.
- The `write` command recalls the factory configuration settings from the modem (the `ATZ` command works with the Macintosh Portable Data Modem 2400). The modem should issue a string in response indicating that it has understood and performed the command.
- The `matchread` command reads the response from the modem and compares it to the match strings placed in the match string list by the `matchstr` command. If the response from the modem is `OK\13\10` execution jumps to label 65 to quit the script. The time parameter with the `matchread` command is measured in tenths of a second (therefore, this `matchread` searches for a match within 3.0 seconds).

Turn off auto-answering

```
@LABEL 64
write "ATS0=0\13"
matchclr
matchstr 1 65 "OK\13\10"
matchread 30
```

When the script exits after hangup, the modem should be reconfigured so that it responds correctly the next time the script is played. This part of the script turns off auto-answering.

- The `write` command disables auto-answering in the modem (check your modem documentation to make sure that `ATS0=0` is the appropriate command). The modem should issue a string in response indicating that it has understood and performed the command.
- The `matchclr` command erases all match strings from the CCL interpreter's match string list.
- The `matchstr` command places `OK\13\10` in the match string list and specifies that control should jump to label 65 if this string is read from the modem.
- The `matchread` command reads the response from the modem and compares it to the match strings placed in the match string list by the `matchstr` command. If the response from the modem matches a string in the match string list, execution jumps to the label of the matching match string. The time parameter with the `matchread` command is measured in tenths of a second (therefore, this `matchread` searches for a match within 3.0 seconds).

Exit

```
@LABEL 65
exit 0
```

This command causes the CCL interpreter to quit the script. The Remote Access program continues with its internal execution. An exit code of `0` means that the script has finished execution without error. Any nonzero error code means that something has gone wrong with the connection.

3 CCL Commands

This chapter describes the CCL commands interpreted by the AppleTalk Remote Access program. Each command section contains a description of the command, the syntax of the command, and an example (if appropriate). The commands are presented in alphabetical order.

! Comment

The `! Comment` statement designates a comment line in the script. The exclamation point is the first character on the command line.

Syntax `! comment`

Example `! Turn echo off`

@ANSWER

The `@ANSWER` label marks the script entry point when the script plays in answer mode.

Syntax `@ANSWER`

@HANGUP

The `@HANGUP` label marks the script entry point when the script plays in hangup mode.

Syntax `@HANGUP`

@LABEL

`@LABEL` sets a numeric label in the script. You can then reference the label from other script commands, such as `JUMP`, `JSR`, and `IFTRIES`. A script may use up to 128 labels, numbered 1 through 128.

Syntax `@LABEL labelnum`

Parameter `labelnum` a value from 1 through 128 that specifies the label number

Example `@LABEL 30`

@ORIGINATE

The `@ORIGINATE` label marks the script entry point when the script plays in originate mode.

Syntax `@ORIGINATE`

ASK

`ASK` causes the CCL to wait for SystemTask (see Volume 1 of *Inside Macintosh* for more information) before displaying a dialog box to obtain information from the user. You may need the `ASK` command if your telephone system uses special telecommunications equipment. This command is used in originate mode only.

Syntax `ASK maskflag "message"`

Parameters

`maskflag`	`1` to mask the user's input with dots ("••••"); `0` to echo the user's input
`message`	the string displayed in the dialog box as a prompt for the user

Example

```
ASK 1 "Enter your password to access the network."
```

CHRDELAY

`CHRDELAY` allows the script to specify a delay time between characters for `WRITE` commands. This is useful for telecommunications equipment that requires data at a slower speed than the interface speed. The delay is inserted between each character of a `WRITE` command.

Syntax `CHRDELAY x`

Parameter

`x`	the delay time, in tenths of a second

Example

```
CHRDELAY 8
```

COMMUNICATINGAT

Use the `COMMUNICATINGAT` command to indicate the speed of the modem connection if the modem speed is different from the serial port speed. This is necessary because the Remote Access application's internal timers are based on the modem speed.

Syntax `COMMUNICATINGAT x`

Parameters

`x`	the modem speed, in bits per second

Example

```
COMMUNICATINGAT 4800
```

DECTRIES

`DECTRIES` decreases the variable tryCounter by one. The CCL interpreter maintains the variable tryCounter, which you may set to a value and increase or decrease by one. Refer also to the commands `IFTRIES`, `INCTRIES`, and `SETTRIES`.

Syntax

```
DECTRIES
```

DTRCLEAR

`DTRCLEAR` clears, or unasserts, the Data Terminal Ready (DTR) signal on the RS-232 interface.

Syntax

```
DTRCLEAR
```

DTRSET

`DTRSET` sets, or asserts, the DTR signal on the RS-232 interface.

Syntax

```
DTRSET
```

EXIT

`EXIT` terminates execution of the script and returns a result along with an optional string. If the script executes successfully, the result returned should be zero. If the script fails for any reason, the result returned is one of the CCL error result codes. For a listing of the CCL error codes, refer to Chapter 4, "Error Codes."

If you want to give the user a nonstandard error, use error code -6002 and use the string parameter to give nonstandard error messages. For example,

```
EXIT -6002 "unable to communicate with PBX"
```

Syntax

```
EXIT result "string"
```

Parameters

`result`	one of the CCL result codes
`string`	the message displayed to the user when a connection attempt fails

The example below shows the script exiting and returning the error result code for a busy signal (`-6022`).

Example

```
EXIT -6022 "The line is busy."
```

FLUSH

`FLUSH` empties all characters from the serial driver input buffer.

Syntax `FLUSH`

HSRESET

`HSRESET` sets the flow control options. For use with AppleTalk Remote Access, you should turn on only CTS hardware handshaking for output. Note that the only option you may need to set is `outputCTS`.

Syntax `HSRESET outputXON/XOFF outputCTS XON XOFF inputXON/XOFF inputDTR`

Parameters

`outputXON/XOFF`	XON/XOFF handshaking for output. This should be off.
`outputCTS`	CTS hardware handshaking for output. This should be on.
`XON`	specifies the XON character (not used)
`XOFF`	specifies the XOFF character (not used)
`inputXON/XOFF`	XON/XOFF handshaking for input. This should be off.
`inputDTR`	DTR hardware handshaking for input. This should be off.

Example

```
HSRESET 0 1 0 0 0 0
```

IFANSWER

If the script is playing in answer mode, `IFANSWER` causes execution to continue at the specified label.

Syntax `IFANSWER jumpLabel`

Parameter `jumpLabel` the label to jump to, where execution continues

Example

```
IFANSWER 30
```

IFORIGINATE

If the script is playing in originate mode, `IFORIGINATE` causes execution to continue at the specified label.

Syntax `IFORIGINATE jumpLabel`

Parameter

`jumpLabel`	the label to jump to, where execution continues

Example `IFORIGINATE 7`

IFSTR

`IFSTR` compares two strings. If the strings are equal, the script continues execution at the specified label.

Syntax `IFSTR varStringIndex jumpLabel "compareString"`

Parameters

`varStringIndex`	the string to compare
`jumpLabel`	the label to jump to, where execution continues
`compareString`	the string with which varStringIndex is compared

In the following example, if the modem speaker flag (varString 2) is 1, execution jumps to label 55. Otherwise, the next command is executed.

Example `IFSTR 2 55 "1"`

IFTRIES

`IFTRIES` compares a parameter with tryCounter. If tryCounter is greater than or equal to the parameter, the script continues execution at the specified jump label. Refer also to the commands `DECTRIES`, `INCTRIES`, and `SETTRIES`.

Syntax `IFTRIES numTries jumpLabel`

Parameters

`numTries`	the parameter to compare with tryCounter
`jumplabel`	the label to jump to, where execution continues

The following example checks to see if tryCounter is greater than or equal to 3. If so, the execution jumps to label 62 and continues.

Example `IFTRIES 3 62`

INCTRIES

`INCTRIES` increases the tryCounter by one. Refer also to the commands `DECTRIES`, `IFTRIES`, and `SETTRIES`.

Syntax `INCTRIES`

JSR

`JSR` causes script execution to jump to the subroutine located at the specific label, saving the address of the line following the `JSR` command. Upon encountering a `RETURN` command, execution resumes at the line following the `JSR` command. `JSR` commands can be nested up to 16 deep.

Syntax `JSR jumpLabel`

Parameter `jumpLabel` the label to jump to, where execution continues

Example `JSR 50`

JUMP

`JUMP` causes script execution to continue at the specified label.

Syntax `JUMP jumpLabel`

Parameter `jumpLabel` the label to jump to, where execution continues

Example `JUMP 59`

LBREAK

`LBREAK` generates a long break (210 tics, 3.5 seconds) on the transmission line.

Syntax `LBREAK`

MATCHCLR

MATCHCLR clears all match strings. The CCL interpreter holds up to 16 strings identified with the MATCHSTR command in a list. The MATCHCLR command is used to erase the contents of the list. You should use MATCHCLR at the beginning of every modem script to clear the list.

Syntax

```
MATCHCLR
```

MATCHREAD

MATCHREAD reads input from the serial driver, and compares the input to the current match strings. If a match is found in the specified time, execution continues at the label of the matching match string.

Syntax

```
MATCHREAD time
```

Parameter

`time` the time allowed for a match, in tenths of a second

The following example searches for a match within 3.0 seconds.

Example

```
MATCHREAD 30
```

MATCHSTR

MATCHSTR specifies a string to match incoming characters against. If a stream of consecutive incoming characters matches the string, script execution continues at the specified label. Sixteen possible match strings exist. Refer to the commands MATCHCLR and MATCHREAD.

Syntax

```
MATCHSTR matchNum matchLabel "matchStr"
```

Parameters

`matchNum`	a value from 1 through 16
`matchLabel`	the label to jump to, where execution continues
`matchStr`	a string (1–255 characters) to compare against

The following example matches the string `"OK\13\10"` with match string 1. If the strings match, then execution jumps to label 8.

Example

```
MATCHSTR 1 8 "OK\13\10"
```

NOTE

The `NOTE` command displays status and log information, passing the message string as a parameter. Optionally, you can set the message level to specify where the message should appear. Message level 1 sends the message to the Activity Log only. Message level 2, which is the default, sends the message to the Remote Access Status window only. Message level 3 sends the message to both the Activity Log and the Remote Access Status window.

The script should log outgoing calls by displaying the dialed phone number in the Status Window and the Activity Log (using `NOTE "Dialing ^1" 3`). Another important status message is the "Communicating at xxxx" message, which lets the user know the speed of the connection.

Syntax `NOTE msgStr msgLevel`

Parameters

`msgStr`	the message parameter passed
`msgLevel`	the message level 1, 2, or 3 (default is 2)

Example

```
NOTE "DIALING ^1" 2
```

PAUSE

`PAUSE` causes script execution to halt for a specified period of time.

Syntax `PAUSE time`

Parameter

`time`	the time to pause, in tenths of a second

The following example causes script execution to pause for 2.0 seconds.

Example

```
PAUSE 20
```

RETURN

`RETURN` terminates a subroutine. Script execution continues with the line following the `JSR` command.

Syntax `RETURN`

SBREAK

`SBREAK` generates a short break (30 tics, 0.5 seconds) on the transmission line.

Syntax `SBREAK`

SERRESET

`SERRESET` configures the serial driver, providing values for baud rate, parity, data bits, and stop bits. Giving a zero value for any of the parameters causes the default value to be used.

Syntax `SERRESET baudRate, parity, dataBits, stopBits`

Parameters

`baudRate`	300, 1200, 2400 (the default), 4800, 9600, 19,200, or 57,600
`parity`	1 for odd parity 2 for even parity 0 or 3 for no parity (the default)
`dataBits`	5, 6, 7, or 8 (the default)
`stopBits`	1 for 1 stop bit (the default) 2 for 2 stop bits 3 for 1.5 stop bits

Example

```
SERRESET 9600, 0, 8, 1
```

◆ **Note** AppleTalk Remote Access requires no parity, 8 data bits, and 1 stop bit. ◆

SETSPEED

`SETSPEED` sets the asynchronous serial interface speed to the specified speed. Use `SETSPEED` to set speeds other than those allowed in `SERRESET`.

Syntax `SETSPEED interfacespeed`

Parameter `interfacespeed` the serial interface speed

Example

```
SETSPEED 14400
```

SETTRIES

SETTRIES initializes the tryCounter variable to the specified value. Refer to the commands DECTRIES, IFTRIES, and INCTRIES.

Syntax SETTRIES tries

Parameter tries the value to assign to the tryCounter variable

Example

```
SETTRIES 0
```

USERHOOK

AppleTalk Remote Access uses the USERHOOK command, with opcode equal to 1, when the script is answering a call and a ring is indicated by the modem.

Syntax USERHOOK opcode

Parameter opcode the parameter passed to the user hook

Example

```
USERHOOK 1
```

WRITE

WRITE writes the specified message to the serial driver.

Syntax WRITE message

Parameter message the message written to the serial driver

The following example sends the message variable string 1 and carriage return to the serial driver.

Example

```
WRITE "ATDT^1\13"
```

4 Error Codes

This chapter lists the error codes returned by the Communication Control Language. Each error code is shown with a description of the error and the message, if any, that is displayed to the user.

Error code	Description	Message displayed
-6002	Generic CCL error.	Supplied by string parameter in the `EXIT` command.
-6003	Subroutine overflow.	The file for the modem you selected does not work properly. It may be damaged; try replacing the file in the Extensions folder.
-6004	The target label is undefined.	The file for the modem you selected does not work properly. It may be damaged; try replacing the file in the Extensions folder.
-6005	Bad parameter error.	The file for the modem you selected does not work properly. It may be damaged; try replacing the file in the Extensions folder.
-6006	Duplicate label error.	The file for the modem you selected does not work properly. It may be damaged; try replacing the file in the Extensions folder.
-6007	Close error.	[No message is displayed.]
-6008	The script was canceled.	[No message is displayed.]
-6009	The script contains too many lines.	The file for the modem you selected does not work properly. It may be damaged; try replacing the file in the Extensions folder.
-6010	The script contains too many characters.	The file for the modem you selected does not work properly. It may be damaged; try replacing the file in the Extensions folder.
-6011	The CCL has not been initialized.	[No message is displayed.]
-6012	Cancel in progress.	[No message is displayed.]
-6013	The `Play` command is in progress.	[No message is displayed.]
-6014	Exit with no error.	[No message is displayed.]
-6015	A label is out of range.	The file for the modem you selected does not work properly. It may be damaged; try replacing the file in the Extensions folder.
-6016	Bad command.	The file for the modem you selected does not work properly. It may be damaged; try replacing the file in the Extensions folder.
-6017	End of script reached; expected `Exit`.	The file for the modem you selected does not work properly. It may be damaged; try replacing the file in the Extensions folder.
-6018	The match string index is out of bounds.	The file for the modem you selected does not work properly. It may be damaged; try replacing the file in the Extensions folder.

(continued)➡

Error code	Description	Message displayed
-6019	Modem error; the modem is not responding.	Modem not responding. Reset modem, check connections, or check to see that the proper port and modem type were specified in the Remote Access Setup control panel.
-6020	No dial tone.	The modem cannot acquire a dial tone.
-6021	No carrier.	The modems could not connect. Try again.
-6022	The line is busy.	The phone number you are calling is busy.
-6023	No answer.	The phone number you are calling does not answer.
-6024	No `@ORIGINATE` command in the modem script.	The file for the modem you selected does not work properly. It may be damaged; try replacing the file in the Extensions folder.
-6025	No `@ANSWER` command in the modem script.	The file for the modem you selected does not work properly. It may be damaged; try replacing the file in the Extensions folder.
-6026	No `@HANGUP` command in the modem script.	The file for the modem you selected does not work properly. It may be damaged; try replacing the file in the Extensions folder.

The Apple Publishing System

The *AppleTalk Remote Access Modem Scripting Guide* was written, edited, and composed on a desktop publishing system, using Apple Macintosh computers, an AppleTalk network system, Microsoft Word, and QuarkXPress. Proof pages were printed on Apple LaserWriter printers. Final pages were printed on a Varityper VT600. PostScript, the LaserWriter page-description language, was developed by Adobe Systems Incorporated.

Text and display type are Apple's corporate font, a condensed version of ITC Garamond®. Bullets are ITC Zapf Dingbats®. Some elements, such as program listings, are set in Apple Courier, a fixed-width font.

APPLE COMPUTER, INC. SOFTWARE LICENSE

PLEASE READ THIS LICENSE CAREFULLY BEFORE USING THE SOFTWARE. BY USING THE SOFTWARE, YOU ARE AGREEING TO BE BOUND BY THE TERMS OF THIS LICENSE. IF YOU DO NOT AGREE TO THE TERMS OF THIS LICENSE, PROMPTLY RETURN THE UNUSED SOFTWARE TO THE PLACE WHERE YOU OBTAINED IT AND YOUR MONEY WILL BE REFUNDED.

1. **License.** The application, demonstration, system and other software accompanying this License, whether on disk, in read only memory, or on any other media (the "Apple Software"), the related documentation and fonts are licensed to you by Apple. You own the disk on which the Apple Software and fonts are recorded but Apple and/or Apple's Licensor(s) retain title to the Apple Software, related documentation and fonts. This License allows you to use the Apple Software and fonts on a single Apple computer and make one copy of the Apple Software and fonts in machine-readable form for backup purposes only. You must reproduce on such copy the Apple copyright notice and any other proprietary legends that were on the original copy of the Apple Software and fonts. You may also transfer all your license rights in the Apple Software and fonts, the backup copy of the Apple Software and fonts, the related documentation and a copy of this License to another party, provided the other party reads and agrees to accept the terms and conditions of this License.

2. **Restrictions.** The Apple Software contains copyrighted material, trade secrets and other proprietary material and in order to protect them you may not decompile, reverse engineer, disassemble or otherwise reduce the Apple Software to a human-perceivable form. You may not modify, network, rent, lease, loan, distribute or create derivative works based upon the Apple Software in whole or in part. You may not electronically transmit the Apple Software from one computer to another or over a network.

3. **Support.** You acknowledge and agree that Apple may not offer any technical support in the use of the Apple Software.

4. **Termination.** This License is effective until terminated. You may terminate this License at any time by destroying the Apple Software, related documentation and fonts and all copies thereof. This License will terminate immediately without notice from Apple if you fail to comply with any provision of this License. Upon termination you must destroy the Apple Software, related documentation and fonts and all copies thereof.

5. **Export Law Assurances.** You agree and certify that neither the Apple Software nor any other technical data received from Apple, nor the direct product thereof, will be exported outside the United States except as authorized and as permitted by the laws and regulations of the United States. If the Apple Software has been rightfully obtained by you outside of the United States, you agree that you will not re-export the Apple Software nor any other technical data received from Apple, nor the direct product thereof, except as permitted by the laws and regulations of the United States and the laws and regulations of the jurisdiction in which you obtained the Apple Software.

6. **Government End Users.** If you are acquiring the Apple Software and fonts on behalf of any unit or agency of the United States Government, the following provisions apply. The Government agrees:

(i) if the Apple Software and fonts are supplied to the Department of Defense (DoD), the Apple Software and fonts are classified as "Commercial Computer Software" and the Government is acquiring only "restricted rights" in the Apple Software, its documentation and fonts as that term is defined in Clause 252.227-7013(c)(1) of the DFARS; and

(ii) if the Apple Software and fonts are supplied to any unit or agency of the United States Government other than DoD, the Government's rights in the Apple Software, its documentation and fonts will be as defined in Clause 52.227-19(c)(2) of the FAR or, in the case of NASA, in Clause 18-52.227-86(d) of the NASA Supplement to the FAR.

7. **Limited Warranty on Media.** Apple warrants the diskettes and/or compact disc on which the Apple Software and fonts are recorded to be free from defects in materials and workmanship under normal use for a period of ninety (90) days from the date of purchase as evidenced by a copy of the receipt. Apple's entire liability and your exclusive remedy will be replacement of the diskettes and/or compact disc not meeting Apple's limited warranty and which is returned to Apple or an Apple authorized representative with a copy of the receipt. Apple will have no responsibility to replace a disk/disc damaged by accident, abuse or misapplication. ANY IMPLIED WARRANTIES ON THE DISKETTES AND/OR COMPACT DISC, INCLUDING THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, ARE LIMITED IN DURATION TO NINETY (90) DAYS FROM THE DATE OF DELIVERY. THIS WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS, AND YOU MAY ALSO HAVE OTHER RIGHTS WHICH VARY BY JURISDICTION.

8. **Disclaimer of Warranty on Apple Software.** You expressly acknowledge and agree that use of the Apple Software and fonts is at your sole risk. The Apple Software, related documentation and fonts are provided "AS IS" and without warranty of any kind and Apple and Apple's Licensor(s) (for the purposes of provisions 8 and 9, Apple and Apple's Licensor(s) shall be collectively referred to as "Apple") EXPRESSLY DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. APPLE DOES NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THE APPLE SOFTWARE WILL MEET YOUR REQUIREMENTS, OR THAT THE OPERATION OF THE APPLE SOFTWARE WILL BE UNINTERRUPTED OR ERROR-FREE, OR THAT DEFECTS IN THE APPLE SOFTWARE AND THE FONTS WILL BE CORRECTED. FURTHERMORE, APPLE DOES NOT WARRANT OR MAKE ANY REPRESENTATIONS REGARDING THE USE OR THE RESULTS OF THE USE OF THE APPLE SOFTWARE AND FONTS OR RELATED DOCUMENTATION IN TERMS OF THEIR CORRECTNESS, ACCURACY, RELIABILITY, OR OTHERWISE. NO ORAL OR WRITTEN INFORMATION OR ADVICE GIVEN BY APPLE OR AN APPLE AUTHORIZED REPRESENTATIVE SHALL CREATE A WARRANTY OR IN ANY WAY INCREASE THE SCOPE OF THIS WARRANTY. SHOULD THE APPLE SOFTWARE PROVE DEFECTIVE, YOU (AND NOT APPLE OR AN APPLE AUTHORIZED REPRESENTATIVE) ASSUME THE ENTIRE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO THE ABOVE EXCLUSION MAY NOT APPLY TO YOU.

9. **Limitation of Liability.** UNDER NO CIRCUMSTANCES INCLUDING NEGLIGENCE, SHALL APPLE BE LIABLE FOR ANY INCIDENTAL, SPECIAL OR CONSEQUENTIAL DAMAGES THAT RESULT FROM THE USE OR INABILITY TO USE THE APPLE SOFTWARE OR RELATED DOCUMENTATION, EVEN IF APPLE OR AN APPLE AUTHORIZED REPRESENTATIVE HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. SOME JURISDICTIONS DO NOT ALLOW THE LIMITATION OR EXCLUSION OF LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES SO THE ABOVE LIMITATION OR EXCLUSION MAY NOT APPLY TO YOU.

In no event shall Apple's total liability to you for all damages, losses, and causes of action (whether in contract, tort (including negligence) or otherwise) exceed the amount paid by you for the Apple Software and fonts.

10. **Controlling Law and Severability.** This License shall be governed by and construed in accordance with the laws of the United States and the State of California, as applied to agreements entered into and to be performed entirely within California between California residents. If for any reason a court of competent jurisdiction finds any provision of this License, or portion thereof, to be unenforceable, that provision of the License shall be enforced to the maximum extent permissible so as to effect the intent of the parties, and the remainder of this License shall continue in full force and effect.

11. **Complete Agreement.** This License constitutes the entire agreement between the parties with respect to the use of the Apple Software, related documentation and fonts, and supersedes all prior or contemporaneous understandings or agreements, written or oral, regarding such subject matter. No amendment to or modification of this License will be binding unless in writing and signed by a duly authorized representative of Apple.

12/19/91
001-0158-B

AppleTalk Remote Access Developer's Toolkit

R0128LL/A

Apple Computer, Inc.
20525 Mariani Avenue, M/S 33-G
Cupertino, CA 95014
(408) 996-1010
TLX 171-576

To reorder products, please call:
1-800-282-2732 (in the United States)
1-800-637-0029 (in Canada)
1-408-562-3910 (International)

AppleTalk Remote Access Protocol (ARAP) External Reference Specifications

APPLE COMPUTER, INC. SOFTWARE LICENSE

PLEASE READ THIS LICENSE CAREFULLY BEFORE USING THE SOFTWARE. BY USING THE SOFTWARE, YOU ARE AGREEING TO BE BOUND BY THE TERMS OF THIS LICENSE. IF YOU DO NOT AGREE TO THE TERMS OF THIS LICENSE, PROMPTLY RETURN THE UNUSED SOFTWARE TO THE PLACE WHERE YOU OBTAINED IT AND YOUR MONEY WILL BE REFUNDED.

1. **License.** The application, demonstration, system and other software accompanying this License, whether on disk, in read only memory, or on any other media (the "Apple Software") and related documentation are licensed to you by Apple. You own the disk on which the Apple Software is recorded but Apple and/or Apple's Licensor(s) retain title to the Apple Software and related documentation. This License allows you to use the Apple Software on a single Apple computer and make one copy of the Apple Software in machine-readable form for backup purposes only. You must reproduce on such copy the Apple copyright notice and any other proprietary legends that were on the original copy of the Apple Software. You may also transfer all your license rights in the Apple Software, the backup copy of the Apple Software, the related documentation and a copy of this License to another party, provided the other party reads and agrees to accept the terms and conditions of this License.

2. **Restrictions.** The Apple Software contains copyrighted material, trade secrets and other proprietary material and in order to protect them you may not decompile, reverse engineer, disassemble or otherwise reduce the Apple Software to a human-perceivable form. You may not modify, network, rent, lease, loan, distribute or create derivative works based upon the Apple Software in whole or in part. You may not electronically transmit the Apple Software from one computer to another or over a network.

3. **Support.** You acknowledge and agree that Apple may not offer any technical support in the use of the Software.

4. **Termination.** This License is effective until terminated. You may terminate this License at any time by destroying the Apple Software and related documentation and all copies thereof. This License will terminate immediately without notice from Apple if you fail to comply with any provision of this License. Upon termination you must destroy the Apple Software and related documentation and all copies thereof.

5. **Export Law Assurances.** You agree and certify that neither the Apple Software nor any other technical data received from Apple, nor the direct product thereof, will be exported outside the United States except as authorized and as permitted by the laws and regulations of the United States.

6. **Government End Users.** If you are acquiring the Apple Software on behalf of any unit or agency of the United States Government, the following provisions apply. The Government agrees:

(i) if the Apple Software is supplied to the Department of Defense (DoD), the Apple Software is classified as "Commercial Computer Software" and the Government is acquiring only "restricted rights" in the Apple Software and its documentation as that term is defined in Clause 252.227-7013(c)(1) of the DFARS; and

(ii) if the Apple Software is supplied to any unit or agency of the United States Government other than DoD, the Government's rights in the Apple Software and its documentation will be as defined in Clause 52.227-19(c)(2) of the FAR or, in the case of NASA, in Clause 18-52.227-86(d) of the NASA Supplement to the FAR.

7. **Limited Warranty on Media.** Apple warrants the disks on which the Apple Software is recorded to be free from defects in materials and workmanship under normal use for a period of ninety (90) days from the date of purchase as evidenced by a copy of the receipt. Apple's entire liability and your exclusive remedy will be replacement of the disk not meeting Apple's limited warranty and which is returned to Apple or an Apple authorized representative with a copy of the receipt. Apple will have no responsibility to replace a disk damaged by accident, abuse or misapplication. ANY IMPLIED WARRANTIES ON THE DISKS, INCLUDING THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, ARE LIMITED IN DURATION TO NINETY (90) DAYS FROM THE DATE OF DELIVERY. THIS WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS, AND YOU MAY ALSO HAVE OTHER RIGHTS WHICH VARY FROM STATE TO STATE.

8. **Disclaimer of Warranty on Apple Software.** You expressly acknowledge and agree that use of the Apple Software is at your sole risk. The Apple Software and related documentation are provided "AS IS" and without warranty of any kind and Apple and Apple's Licensor(s) (for the purposes of provisions 8 and 9, Apple and Apple's Licensor(s) shall be collectively referred to as "Apple") EXPRESSLY DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. APPLE DOES NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THE APPLE SOFTWARE WILL MEET YOUR REQUIREMENTS, OR THAT THE OPERATION OF THE APPLE SOFTWARE WILL BE UNINTERRUPTED OR ERROR-FREE, OR THAT DEFECTS IN THE APPLE SOFTWARE WILL BE CORRECTED. FURTHERMORE, APPLE DOES NOT WARRANT OR MAKE ANY REPRESENTATIONS REGARDING THE USE OR THE RESULTS OF THE USE OF THE APPLE SOFTWARE OR RELATED DOCUMENTATION IN TERMS OF THEIR CORRECTNESS, ACCURACY, RELIABILITY, OR OTHERWISE. NO ORAL OR WRITTEN INFORMATION OR ADVICE GIVEN BY APPLE OR AN APPLE AUTHORIZED REPRESENTATIVE SHALL CREATE A WARRANTY OR IN ANY WAY INCREASE THE SCOPE OF THIS WARRANTY. SHOULD THE APPLE SOFTWARE PROVE DEFECTIVE, YOU (AND NOT APPLE OR AN APPLE AUTHORIZED REPRESENTATIVE) ASSUME THE ENTIRE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION. SOME STATES DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO THE ABOVE EXCLUSION MAY NOT APPLY TO YOU.

9. **Limitation of Liability.** UNDER NO CIRCUMSTANCES INCLUDING NEGLIGENCE, SHALL APPLE BE LIABLE FOR ANY INCIDENTAL, SPECIAL OR CONSEQUENTIAL DAMAGES THAT RESULT FROM THE USE OR INABILITY TO USE THE APPLE SOFTWARE OR RELATED DOCUMENTATION, EVEN IF APPLE OR AN APPLE AUTHORIZED REPRESENTATIVE HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. SOME STATES DO NOT ALLOW THE LIMITATION OR EXCLUSION OF LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES SO THE ABOVE LIMITATION OR EXCLUSION MAY NOT APPLY TO YOU.

In no event shall Apple's total liability to you for all damages, losses, and causes of action (whether in contract, tort (including negligence) or otherwise) exceed the amount paid by you for the Apple Software.

10. **Controlling Law and Severability.** This License shall be governed by and construed in accordance with the laws of the United States and the State of California, as applied to agreements entered into and to be performed entirely within California between California residents. If for any reason a court of competent jurisdiction finds any provision of this License, or portion thereof, to be unenforceable, that provision of the License shall be enforced to the maximum extent permissible so as to effect the intent of the parties, and the remainder of this License shall continue in full force and effect.

11. **Complete Agreement.** This License constitutes the entire agreement between the parties with respect to the use of the Apple Software and related documentation, and supersedes all prior or contemporaneous understandings or agreements, written or oral, regarding such subject matter. No amendment to or modification of this License will be binding unless in writing and signed by a duly authorized representative of Apple.

7/15/91
001-0158-A

AppleTalk® Remote Access Protocol (ARAP) External Reference Specifications

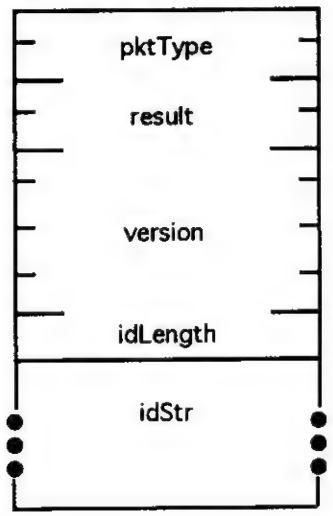
These are the specifications for the underlying protocols currently being used in the AppleTalk Remote Access product version 1.0.

***NOTE**: Although every attempt has been made to verify the accuracy of the information presented, this document may contain errors and is subject to change.*

Apple Computer, Inc.

After the PPL is established, the originating (or calling) side of the connection sends a Connection Request (CR) packet to the accepting (or answering) side of the connection. The acceptor directs the CR to the appropriate service based upon the ID string contained in the CR. The answer side of the connection can accept or refuse the connect attempt, and indicates the result by sending the Connection Request Result packet to the Originator. The format of the CR and CRR packets is shown below.

Link Arbitration Packet



The pktType field indicates the type of the packet. The value 1 indicates the packet is a Connect Request, a 2 indicates the packet is a Connect Request Result. The result field is zero in the CR, and indicates the result of the connection request in the CRR packet. A zero value for the result in the CRR packet indicates that the Connection Request was successful and ARAP can use the link. A non-zero value indicates the CR failed. Possible error codes include the following:

-5902	Service requested is not available
-5906	Incompatible arbitration packet version
-5915	Connection refused by acceptor

The version field indicates the version of the arbitration packet format. The version is a four byte version defined by Apple two have the following parts (packed into a long in order): 1) First part of the version number in BCD. 2) Second and third parts. 3) Release type (development=0x20, alpha=0x40, beta=0x60, release=0x80). 4) Stage of prerelease version. If the acceptor does not support the callers version, a -5906 result is returned. The idStr is a Pascal style string which indicates the service the caller wants to connect to. The ARAP idStr value is "Remote Access". The idLength is the length of the idStr, including the length byte.

ARAP POINT TO POINT LINK

The AppleTalk Remote Access Protocol (ARAP) runs on top of a Point to Point Link (PPL). This document describes the characteristics of the PPL, the link arbitration protocol, and the Modem Link Tool. The Modem Link Tool provides a PPL for ARAP over telephone lines. Many of the details of the PPL are dependent on the particular implementation.

Point to Point Link Characteristics

In order to work with ARAP, each PPL must support the following basic set of characteristics:

- A connection for ARAP to send and receive "packets". ARAP expects a packet (i.e. block) type interface to the PPL. In other words, the data contained in one write request to the link, must be delivered to ARAP as the result of one read request. The framing of the packets is implementation dependent, but must relay the packet size information.

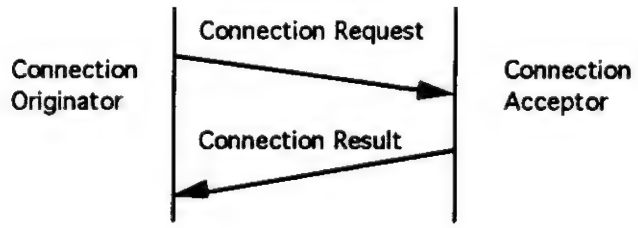
- A packet size of at least 604 bytes.

- Best effort, point to point data delivery. ARAP depends on the PPL to detect data errors in the packets, and to discard packets that contain errors. All data delivered to ARAP must be valid data sent by the other side of the link.

- Optionally, the PPL can provide reliable link. A reliable link guarantees that packets are delivered in the same order they are sent, and are free of duplicates. If the link is reliable, ARAP provides some enhancements which can greatly reduce the amount of redundant traffic sent across the link.

Link Arbitration Protocol

It is expected that the PPL used by ARAP may be shared other services. For example, a phone line and modem may be set up to receive ARAP connections as well as electronic mail connections. For this reason, when ARAP is establishing a connection, the side originating the connection arbitrates for the use of the link with the answering side of the connection. The arbitration dialog is as follows:



APPLETALK REMOTE ACCESS PROTOCOL

The AppleTalk Remote Access Protocol (ARAP) provides efficient AppleTalk services on a per client basis over slow links. It defines the login and authentication sequence as well as the Appletalk data format. There are two broad types of ARAP information: 1) Internal messages (authentication packets, tickle packets, etc.) that define information related to establishing and maintaining the link. 2) AppleTalk packets which contain the higher level (DDP and above) data that is to be sent. This document describes version 1 of ARAP.

If the point to point link that we are operating on can ensure reliable, in order, delivery of data we can provide some optional enchancements to the AppleTalk packet delivery mechanism. These enhancements are very desirable since they can greatly reduce the amount of redundant traffic sent across the link.

Internal Messages

In order to operate over a wide range of point to point links we must not require that the underlying link provides reliable packet delivery for proper operation of this protocol. Therefore, for internal messages we have defined a very simple protocol to deliver the information sent in a reliable fashion. This protocol is not tailored to any specific link and is limited to sending one message at a time. This is sufficient for our needs since we send a very low volume of internal messages relative to the amount of AppleTalk information. It is assumed that the underlying link provides the framing of the packet and has a way to relay the size of the data sent. We also assume that the underlying link can support packets of at least 604 bytes in size. NOTE: All numbers passed in internal message packets are stored big endian.

DataGroup Flag:

This flag always precedes a datagroup and is used to describe the information that follows. It is a one byte value and is broken down into the following fields:

Bit 7: Reserved -
This bit must be set to zero.

Bit 6: Packet Data -
This bit is set if the data in this packet is data. It is clear if it is an out of band message.

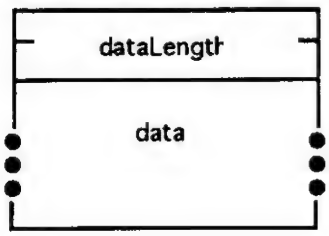
Bit 5: Tokenized -
This bit is set if the data after this flag is a token. If it is a token, the sequence number of the token is in the word that follows (always a word, not a compactnum). If this bit is not set, it is assumed that raw data follows. The raw data is preceded by a compacted length value unless it is the last group of data in the packet. The last group of raw data in a packet is a special case in which the length can be derived from the amount of data remaining in the packet.

Bit 4:Last Group -
This bit is set if the data that follows is the last group of data in the packet. This flag must be present in the last datagroup. If raw data is being described and this flag is set, the size of the data is calculated as specified above.

The Modem Link Tool

The Modem Link Tool (MLT) provides a **reliable** PPL over asynchronous serial connections. The MLT has two major components. The first component, the CCL, establishes the physical connection (e.g. connects the modems). After the physical connection is established, the second component, the reliable protocol, provides a reliable PPL. The reliable protocol used is the V.42 Alternative Procedure, with V.42bis data compression. The protocol is described in ANNEX A of the V.42 specification. On top of the reliable protocol, the MLT uses a simple packet format to ensure packet delivery for ARAP. The MLT packet format used on top of the reliable protocol is a follows:

Modem Link Tool Packet



The dataLength field is the length of the data in the packet, and does not included the length bytes. The maximum dataLength supported by the MLT is 618 bytes.

Initializing SRP

Both sides of the connection must start off by initializing their outgoing and incoming (expected) sequence numbers to zero. This allows both sides to start off in sync with each other.

Writing data

In order to write a packet using SRP, you set bit 6 of the flag byte to zero, the command byte to the command wanted, and the sequence number to the current outgoing sequence number. The rest of the packet is made up of the data that is represented by the command we are sending.

After the packet is sent, we must receive an ack that matches the sequence number we sent to indicate successful delivery of the packet. If after a period of n seconds (where n=6 for 1200 bps, n=3 for 2400 bps, n=2 for 4800+ bps) no ack has been received, we send the packet again. This process will take place until the ack is received or a period of 60 seconds elapses in which case the connection is torn down with a timeout error.

Once the ack is received successfully we increment our current outgoing sequence number by one for the next write. Since this is a one byte value we wrap back to zero after we add 1 to a sequence number of 255.

Reading data

In order to read data we must first determine if an incoming packet is an internal message. We do this by checking the bit 6 of the flag byte to see if it is zero. If it is, we have an internal message packet.

The incoming sequence number is then checked to see if it equals the expected incoming sequence number. If it is not the sequence number expected, an ack should be sent back if the sequence number is one less than expected. The ack should be returned with the received sequence number. This could happen if we receive a duplicate of a message we have already accepted. If the packet matches the expected sequence number an ack is sent, the expected sequence number is bumped, and the data is delivered.

ARAP Connection Establishment

After the low level link has been established we begin the process of establishing the ARAP link. In this discussion the client represents the originator, and the server represents the answering or accepting side of the connection. Before reading or writing any data both sides initialize the SRP. The sequence of events in connection establishment is critical and must be followed exactly.

Bit 3: Range -
This bit indicates whether or not the sequence number that follows describes a range or not. If the bit is set, the sequence number is describing part of a sequenced packet, not the entire packet. The bytes that are wanted from that packet are described by the two compact numbers that follow the sequence number word.

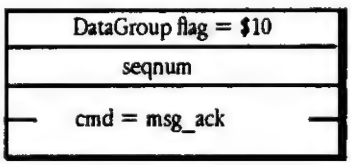
Bit 2: Want Sequenced -
This bit indicates whether or not the group of data described by this flag should be sequenced and cached by the receiver. If the bit is set, the receiver must cache and sequence the data described in this datagroup. The data described can be either raw data or data described by a sequence. If it is data described by a sequence it must be decoded before the entry is made into the cache. If the bit is clear, no entry is made for this datagroup in the receivers cache.

Bit 1: Want Packet Sequenced -
This bit only applies to the first flag in the packet. If it is set, the entire packet should first be decoded and then entered into the receivers cache. It is important that the packet be scanned and any entries requested within the packet be created before the packet itself is sequenced.

Bits 0: Reserved
Must be zero.

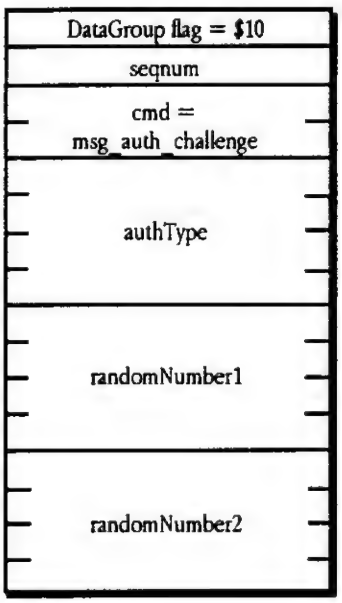
Simple Reliable Protocol (SRP)

All packets sent are preceded by a flag byte that indicates whether this packet is an internal message or an AppleTalk packet. If you are sending an internal message set bit 6 of the flag byte to zero. All internal message packets are then followed by a one byte packet sequence number and a two byte command. The sequence number is used to ensure the ordering of packets that have been sent. The command defines the type of data that this packet represents. The ack command (cmd=0) has a special meaning. It does not define any data but is sent to acknowledge the arrival of a message from the other end of the link.



Server-

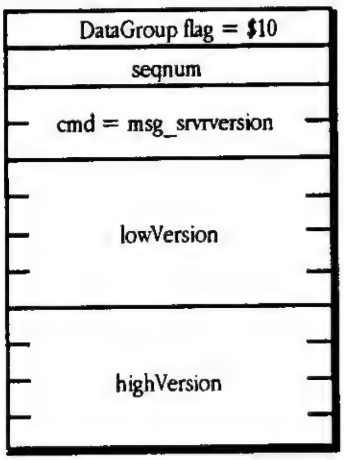
Reads message and confirms that it is msg_rmtversion and that the result is zero. If it is not, the session is torn down. If it is, we check the version sent to us to make sure that it falls within our acceptable range of versions. If it does not we tear the session down. If it does, we know which version of the protocol to use and continue by creating an authentication challenge packet. This packet contains a authType field which represents the type of authentication being done. Currently the only authtype used is two way DES (authtype=kAuth_TwoWayDES). In the two way DES packet, two 32 bit random numbers are generated and put into randomNumber1 and randomNumber2. The packet is then sent (cmd=msg_auth_challenge).



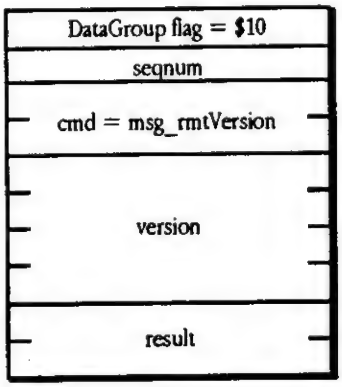
Client-

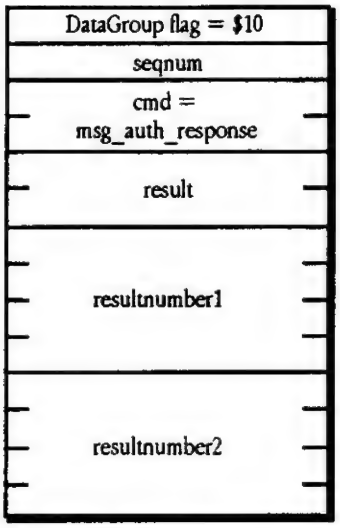
Reads message and confirms that it is msg_auth_challenge and that authType equals kAuth_TwoWayDES. If not, result is set to ERR_VLD8_BADVERSION and a reply is sent (cmd=msg_auth_request). The session is then torn down. If the message is ok, we form an authentication request packet. If we are logging in as a guest, we set the guest flag and set the username to "<Guest>". If we are logging in as an authenticated user, we set the userName to our user. The field userName is an array of 34 characters. If the string you are returning in userName is less than 34 characters the remaining space needs to be padded. If we are authenticating ourselves to the server we must use the randomNumber1 & randomNumber2 sent by the server produce a resulting number. The technique used is to copy the password (excluding the len byte) into a space of 8 bytes. Any unused bytes are set to zero. A key is then generated from the password. This key is then used to encode the 64 bits of random number information passed to us by the server. The resulting number is put into our outgoing packet in resultNumber1 and resultNumber2. We then generate our own random number to challenge the server (we want to be sure it really knows our password) and store it in randomNumber1 and randomNumber2. Finally the result is set to zero and the packet is sent

Server-
Writes a message (cmd=msg_srvrversion) that has the lowest and highest versions that the server can accept. The version is a four byte version defined by Apple to have the following parts (packed into a long in order): 1) First part of the version number in BCD. 2) Second and third parts. 3) Release type (development=0x20, alpha=0x40, beta=0x60, release=0x80). 4) Stage of prerelease version.



Client-
Reads message and confirms that it is msg_srvrversion. If it is not a msg_srvrversion, a reply is sent (cmd=msg_rmtversion) with a result code of ERR_VLD8_BADVERSION, and the session is torn down. If it is msg_srvrversion, we attempt to find a suitable version by taking the minimum of the highVersion from the server and the highest version we support. This gives us the maximum version acceptable to both sides. We then check to see if this version is less than our minimum version supported or less than the lowVersion from the server. If it is, we do not have an acceptable version match with the server and must send a reply (cmd=msg_rmtversion) with an result code of ERR_VLD8_BADVERSION, and tear down the session. If the version is acceptable we set version=acceptableversion and send it (cmd=msg_rmtversion) to the server with an result code of zero.

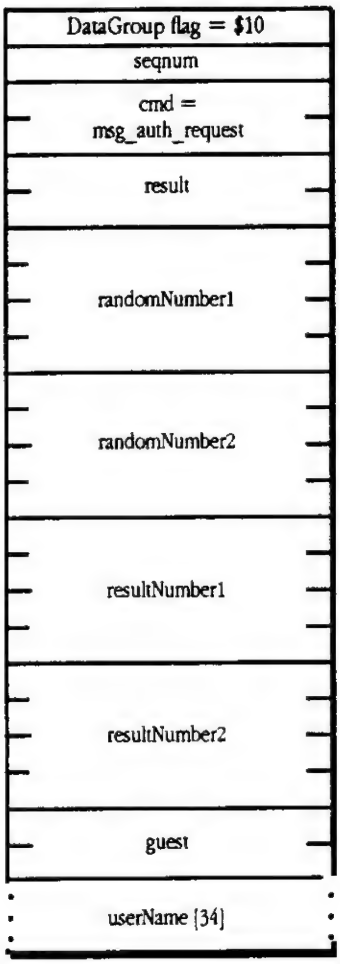




Client-
Reads message and confirms that it is msg_auth_response. If the result is not ERR_VLD8_CALLBACK or zero, we tear down the connection. If we are not doing guest login we check the resultNumbers to make sure they are expected. If not, the connection is torn down. If the result is zero we continue . If the result is ERR_VLD8_CALLBACK, we disconnect, initialize SRP and wait to answer.

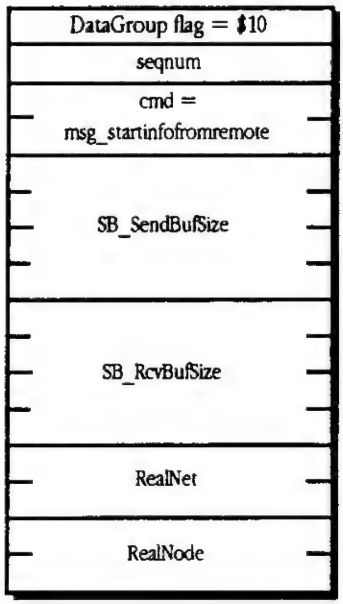
Server-
The server puts together a packet that has information about itself and how the client should appear. RealNet and RealNode are set to the address of the server itself. This information is used in clients that use remapping to ensure that they will be able to communicate with the server under all conditions. AppearAsNet and AppearAsNode defines the ddp network address that the client should use when creating AppleTalk packets. Since the AppearAsNode field is defined as an unsigned short, the high byte of this field will always be zeroed. SB_SendBufSize and SB_RcvBufSize define the amount of space allocated to SmartBuffering for sending and receiving respectively. If SmartBuffering is not used you should set both of these to zero. SmartBuffering can only be used if the underlying link is known to provide reliable delivery of packets. MaxConnectTimer defines the maximum amount of time in seconds that this client is allowed to stay connected. This value will be -1 if the time is unlimited. ServersZone defines the zone that the client will appear in. ServersName should be set to the name of this server. This string appears to the user as the name of the computer that they are connected to. The fields serverZone and serversName are arrays of 34 characters. If the string you are returning in serverZone and serversName are less than 34 characters the remaining space needs to be padded. After these fields are filled in, the packet is sent (cmd=msg_startinfofromserver).

(cmd=msg_auth_request).

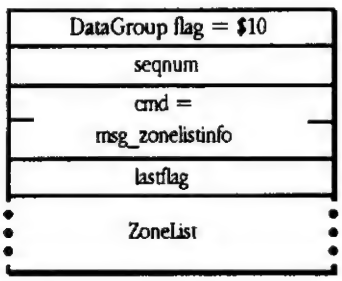


Server-

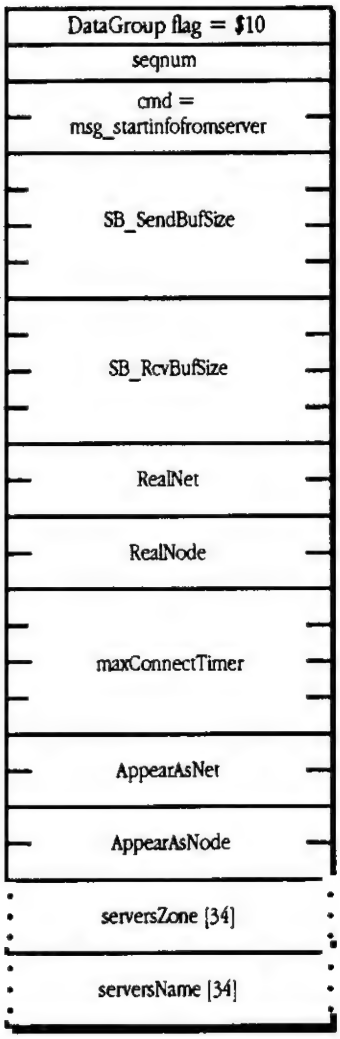
Reads message and confirms that it is msg_auth_request and that the result is zero. If it is not, the session is torn down. If it is, we confirm that the userName and resultNumber returned to us is valid. If they are trying to log in as a guest and we don't support guest the error is ERR_VLD8_GUESTNOTALLOWED. If it is a bad user we set the error to ERR_VLD8_BADUSER. If the resultNumber does not match we set the error to ERR_VLD8_BADPASSWORD. If we got an error we send it back (cmd=msg_auth_response) in result. If everything is ok and we are not doing guest login, we generate a resultNumber the same way the client did above to authenticate to it that we really know the password. At this stage it is possible that this user should be called back. If this is the case we set a result code of ERR_VLD8_CALLBACK to indicate that we will be doing callback. If the user was authenticated and no callback was wanted, we set the result code to zero. We then send the message back to the client (cmd=msg_auth_response). If we are doing callback, we disconnect, initialize SRP, and attempt to connect to the client through callback.



Server-
Reads message and confirms that it is msg_startinfofromremote. If not the connection is torn down. In order to determine the proper size of the buffers for SmartBuffering, the minimum of the clients buffer sizes and our own is obtained. Next, we begin sending the list of allowable zones to the client (cmd=msg_zonelistinfo). This list of pascal strings must be sorted in ascending order. Since this list may be larger than what can fit into one packet we break it into multiple packets. Only the last packet should set the lastflag to true. Care must be taken to ensure that only complete zone names are fit into a packet (we do not allow part of a zone name in one packet and the other in the next). After all zones are sent the connection is established and we can now accept and send AppleTalk packets.



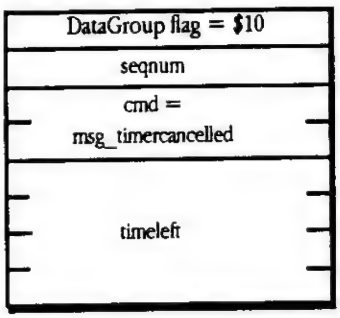
Client-
Reads message and confirms that it is msg_zonelistinfo. If not, connection is torn down. Packets are read until the lastflag is true. The connection is established and we can now send and receive AppleTalk packets.



Client-

Reads message and confirms that it is msg_startinfofromserver. If not, the connection is torn down. In order to determine the proper size of the buffers for SmartBuffering, the minimum of the servers buffer sizes and our own is obtained. If SmartBuffering is not being used we set our SB_SendBufSize and SB_RcvBufSize to zero. SmartBuffering can only be used if the underlying link is known to provide reliable delivery of packets. As a courtesy to the server we indicate our real address (this will be different than the appearAs address if we are using remapping) in RealNet and RealNode. The packet is then sent (cmd=msg_startinfofromremote).

Timer Cancelled-
This message (cmd=msg_timercancelled) indicates that a time left message is being cancelled and the connection is no longer being shut down. The time left indicates the new time left in the connection (generally -1 to indicate unlimited).



AppleTalk Packets

Once the connection has been established we can start to send AppleTalk packets across the link. The format of the packets depends on whether they are being encapsulated in SmartBuffering or not. If SmartBuffering is not being used, the flag byte that precedes the Appletalk data has the bits sflag_PktData and sflag_LastGroup set. This indicates that the packet represents Appletalk data and that the Appletalk data is the last group of data in this packet.

To provide a consistent state (and to improve SmartBuffering) we set certain fields to a known value before transmitting a packet. All packets are sent using the long ddp form. Both the lap source and lap destination bytes are set to zero. If a checksum has been set in the packet, the lap source and lap destination bytes are set to 1. This indicates to the receiver that it must recalculate the checksum. The length byte of ddp is also set to zero. This must be recalculated by the receiver before delivering a packet or forwarding it onto the net.

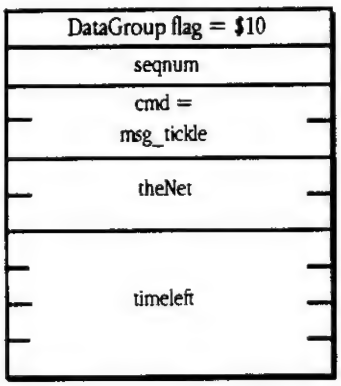
Client responsibilities
When the client needs to send an NBP lookup it should set the nbp function to nbpBrRq, even if there is no router. This allows the server to distinguish whether or not this packet represents a NBP lookup or an NBP confirm. Confirms will always have their nbp function set to nbpLkUp.

Server responsibilities
The server is responsible for properly forwarding packets sent by the client. If an NBP packet arrives with a function code set to nbpBrRq, it treats the lookup as if it had originated from its own stack as described in Inside AppleTalk. This means it must determine whether to forward this lookup to a router if there is one or to send it on its own net. If it is on an extended net, it must also check to see if this lookup is for zone '*'. If it is, it must expand it to its zone if it has one before sending onto the net.

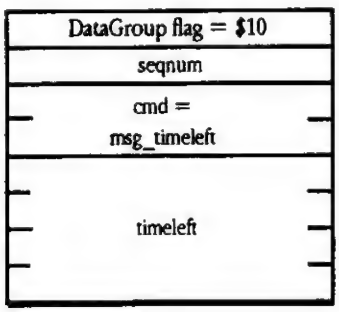
Session maintenance packets-

The following messages can be sent any time after the connection has been established.

Tickle-
This message (cmd=msg_tickle) informs the other side that the connection is still intact. It contains within it the network number of the sender (so the client can adapt to a change from net zero on nonextended nets). It also contains the time left in this session in seconds (only set by the server). A value of -1 indicates that there is unlimited time remaining in the connection. A tickle packet should be sent every 20 seconds and if no valid packets (data or internal message) are received within 60 seconds, the connection is torn down. The reception of any valid packet will cause us to reset our teardown timer. If neither theNet or timeleft has changed, and a valid packet has been received in the last 20 seconds, then no tickle packet needs to be sent.



Time Left-
This message (cmd=msg_timeleft) is sent by the server to inform the client that it only has the number of seconds in timeleft remaining. This message should generally be dealt with by informing the clients user that his connection will be torn down in timeleft seconds.



Error codes:

ERR_VLD8_CALLBACK	-5819
ERR_VLD8_BADVERSION	-5820
ERR_VLD8_BADUSER	-5821
ERR_VLD8_BADPASSWORD	-5822
ERR_VLD8_BADLINK	-5823
ERR_VLD8_NOCALLBACKALLOWED	-5824
ERR_VLD8_ALLCBSERVERSBUSY	-5825
ERR_VLD8_GUESTNOTALLOWED	-5826
ERR_VLD8_SERVERISIMPOSTER	-5827
ERR_VLD8_LOGINNOTENABLED	-5828

Message numbers:

msg_ack	0
msg_srvrversion	1
msg_rmtversion	2
msg_auth_challenge	3
msg_auth_request	4
msg_auth_response	5
msg_startinfofromserver	6
msg_startinfofromremote	7
msg_zonelistinfo	8
msg_tickle	9
msg_timeleft	10
msg_timercancelled	11

Flag bit masks:

sflag_Fixup	0x80
sflag_PktData	0x40
sflag_Tokenized	0x20
sflag_LastGroup	0x10
sflag_Range	0x08
sflag_WantSeqd	0x04
sflag_WantPktSeqd	0x02
sflag_Reserved	0x01

Authentication types:

kAuth_TwoWayDES	1

The server is also responsible for keeping packets originated by the client from being returned to the client. This includes both packets that have a source address equal to the client's address in the DDP header as well as NBP packets that have a source equal to the client. For example, if the client sends a packet and the server forwards this packet to a router, the router may generate a lookup request. It is then the reponsibility of the server to keep the lookup request from returning to the client.

The server is also responsible for eliminating broadcast RTMP information from being forwarded to the client. This information is not needed by the client since the client is simply just another node on the servers net.

```
typedef struct TPacketData
{
        long                    right;          /* link for receive, cksumhead for send */
        long                    left;           /* link for receive */
        unsigned short seqnum;/* sequence number of this entry */
        unsigned short datalen;         /* length of data that follows */
        unsigned char           databuf[];      /* data that follows is tacked on here */
} TPacketData, *TPPacketData;
```

Any block of data added to the ring buffer must always fit without wrapping. In other words, if there is not enough room at the high end of the buffer to store the full block of data, the buffer must be wrapped back to the low end. This is done to simplify the mapping of data structures onto the data within the buffer (at the expense of slightly under utilizing the available space). It would be very cumbersome to deal with the data if part of it was in one place and the other part somewhere else. If there is not enough room to insert an entry, items are removed until enough space exists. It is important to remember that both the sender and the receiver must use exactly the same approach to manage their buffers because they must always stay in sync.

In our implementation, when used for sending, the right field is used to link together the checksum records that describe this block of data. It is not a requirement that this field be used in this way, it is only a requirement that the fields specified in the header exist.

The size of the entire record which includes the actual data must always be an even number of bytes. This is to ensure that subsequent entries will always have their headers aligned on even boundaries. Therefore, if the actual size of the record happens to be an odd number, one byte is added to the amount of space used in the cache.

In order to describe the data in this buffer in a tokenized format, a sequence number is stored with the data. This sequence number will be duplicated on the receive side when it stores this data. It is crucial that the sender and receiver use the same technique to generate sequence numbers to ensure that both sides stay in sync. In order to do this, both sides always start at a sequence number of zero before any data is stored. For each new item inserted into the data buffer, a new sequence number is generated by adding one to the previous one.

When adding data to the buffer, the following technique must be followed by both sides in order for them to stay in sync. The packet is scanned and all operations are executed in order. After all of the partial data packets are added, the entire packet may be added if the appropriate bit has been set in the first datagroup flag byte. Adding the entire packet last ensures that the receiver will be able to reconstruct the packet based only on previous information. If the packet was added first, it would be possible to have entries in the data that referenced the packet itself. If this were allowed to happen, the receiver would have no way of reconstructing the packet since the packet itself would be required!

Each packet always contains at least one datagroup. It may contain more than one datagroup if required. Each datagroup is made up of a flag followed by data whose meaning can be discovered by decoding the flag. A datagroup is a subunit of the packet that can be used to mix different types of data. For example, it might be desirable to describe a packet by using both matched data using sequence numbers as well as new data that is sent in its raw format.

SMARTBUFFERING

SmartBuffering is an algorithm that is used to reduce repetitive traffic such as packets that are sent on a network. It can be tailored to recognize entire or partial packets of information. It works by caching a specified amount of information and checking future packets against that information for matches. If matches are found, a token representing that data is put into the data stream instead of the actual data. Smartbuffering provides the ability to mix both tokens and raw data in the data stream.

Smartbuffering is a state driven algorithm. It requires that both the sending and the receiving side always be in sync. Therefore, the medium used to transmit its data must provide a reliable point to point link. It also requires that the sender and receiver use the same size buffers in order to remain synced up when new entries items begin overflowing into earlier ones. Because of this it is necessary to exchange buffer sizes before any data is processed. The buffer size that is used is the smaller of the two buffers to ensure that both sides end up with the same size buffers since the side with the larger buffer can always temporarily reduce the amount of buffer it provides.

The choice of what data to cache and how to recognize it when matching subsequent packets is entirely that of the sender. Smartbuffering describes the format of the packets used to transmit the data and how the actual data is stored. It does not explicitly describe how matches are found on the sending side or how any indexes into the send side data are stored. Determining what to match and how to match the data is implementation independent and can be tailored for the needs at hand.

Implementation Details

At the heart of Smartbuffering lies the techniques used to ensure that the sender and receiver always remain in sync.

The first step in ensuring this is to provide a method in which both sides can determine the proper size buffer to use. In order to do this they exchange their preferred buffer sizes. After getting the other's buffer size each chooses an actual buffer size that is the smaller of the two. Thus, if the sender had 10000 bytes of storage and the receiver had 8000 bytes of storage the resulting size would be the smaller of the two which is 8000 bytes. This ensures that every operation done to the sender's buffer will result in exactly the same buffer state on the receiving side after the same operation is carried out.

The second step to ensuring that both sides remain in sync is for them to provide a consistent way of adding to and removing items from the data cache. The algorithm used in Smartbuffering uses a ring buffer. The method of storing data in this buffer must be exactly the same on both sides. Each packet inserted into the buffer has the following format:

token we instruct the receiver to enter the data described by this range into its cache. This allows us to provide a full match on the header if we get another packet with the same header. Once we have told the receiver to buffer the full header, we can describe it without using a range thus saving the extra bytes that would have been needed. Since ddp headers are quite similar (most transactions end up going to the same places) we have typically seen the entire header of any packet reduced down to 3 bytes. This honing down technique is used in most of the other header interpretations.

NBP data -
NBP packets tend to be very similar. The most likely items to change are the function and nbpid fields. Fortunately they are situated next to each other. Since it is very likely that the data that follows will produce a match in a later packet we cache it using the honing down technique described in the DDP Header. This produces a high likelihood of a match and allows us to describe it in a small number of bytes. We have seen between 5 and 20 to 1 compression of the nbp traffic.

ATP data-
The entire ATP header is skipped and only the data is searched for a match. Since the ATP Header is only 8 bytes we would see minimal return for special casing that part of the packet. The data however could produce substantial reduction if retransmissions with different headers were occurring.

ADSP data-
Since the ADSP header is quite likely to change from one packet to the next, it is ignored. Again, the data beyond the header is matched since a retransmission could produce a large savings even if the header had changed.

Other data-
For types of packets that we do not special case we simply take all of the data following the DDP header and attempt to match it. As new data types become known, it is possible that they will be added to our list of special cases.

In the future it is also possible that we may take advantage of the sflag_Fixup bit to special case even more details of the packet. This mechanism has not been completely defined by SmartBuffering at this time.

After the special casing of the packet is done we create an entry for the entire packet. This gives us the potential of finding entire packet matches in the future. It is possible through this mechanism to achieve 200 to 1 compression on packets that are exact duplicates of some other packet that preceded it.

In order to minimize the amount of data required to describe the information that follows the datagroup flag we have defined a compact representation of an unsigned short number. This compact representation is known as a compactnum and has the following characteristics: 1) If the high bit of the first byte (first is described as reading left to right within the packet) is set, then we mask off the high order bit to obtain the number. This means that if the value wanted is less than 128 it can be described in 1 byte. If the bit is not set, then the number is a two byte number, and the low order value of the number is obtained from the next byte. This numbering scheme limits the maximum value of a compactnum to 32767.

Redundant traffic reduction

If the link that we are running over ensures reliable packet delivery and the two ends of the connection have negotiated for SmartBuffering, we can take steps to reduce much of the redundant traffic between the two points. SmartBuffering interprets AppleTalk packets based on protocol type, and then creates checksums that define the parts of packets that are most likely to be repetitive.

The receive side of our implementation uses the standard SmartBuffering techniques. It simply obeys the requests of the send side to reproduce the data. Therefore, we will not discuss any details of the receive side.

The effectiveness of SmartBuffering depends on a number of factors. One of the most important factors is how often recognizable packet components match. A good example of highly repetitive data is NBP. Typically the same (or very similar) packets are sent a number of times. Therefore NBP data achieves very good compression since it is so repetitive (typically better than 5 to 1). Another factor in the effectiveness of SmartBuffering is how large the actual history buffers are. Obviously, the larger the buffers the more likely we are to find a match with some information that was previously sent. In our current implementation we use a send an receive history buffer size of 11200.

When a packet is ready to be sent, specific checks are made depending on the type of packet.

First, all packets are checked to see if the entire packet matches (the only exception to this are echo packets which we always want represented in their full form). If we get a full packet match, then the token that describes that match is sent to the receive side. No more interpretation is done, since the partial interpretations of that packet would already have been done and are probably still in the data cache.

If an entire match of the packet cannot be made we take special action depending on the type of data within the packet.

DDP Header -
ARAP always creates long headers when sending packets. Therefore, we only need to interpret long headers. The first time we see a header, a checksum entry is created that describes the data that includes all of the lap and ddp part of the header. If a subsequent packet has a header that matches one that has already been checksummed we substitute the raw data with that of a datagroup token range. Since a range takes between 2 and 4 more bytes to describe than a complete

of the packet (sequence number 2). Finally, the entire packet is entered as sequence number 3. The following data results:

```
DataGroup Flag     -> 6E (PktData, Tokenized, Range, WantSeqd, WantPktSeqd)
Sequence Number    -> 00 00
Range              -> 80 8f (compact numbers for 00 0f)
DataGroup Flag     -> 40 (PktData)
Raw Data Len       -> 82 (compact number for 02)
NBP Func & ID      -> 21 75 (the 2 bytes of raw data)
DataGroup Flag     -> 7C (PktData, Tokenized, LastGroup, Range, WantSeqd)
Sequence Number    -> 00 00
Range              -> 92 A4 (compact numbers for 12 24)
```

As you can see, the resulting packet is 14 bytes long, and the original packet was 37 bytes long, for a savings of 23 bytes. If yet another NBP Lookup is sent and the NBP ID byte changes once again we get the following packet to process:

```
Third Lookup Packet:

LAP/DDP Header     -> 00 00 02 00 00 00 00 11 11 22 22 11 22 02 FE 02
NBP Func & ID      -> 21 76 (Note the ID change from 75)
NBP Tuple          -> 22 22 22 FE 00 01 = 09 AFPServer 01 *
```

The transmitter will once again go through the process of attempting to match the packet. It will not find the entire packet match since the NBP ID changed again. It will find a match for the LAP/DDP Header, but this time it will find the match in sequence number 1 (the sequence it created for this portion of the packet in the above transmission). The advantage this time is that this sequence exactly describes the part of the packet it is looking for, and will be able to describe it without the range bytes. We then output a raw data description for the NBP Function and ID bytes. Then we come to the NBP tuple which is matched with sequence number 2. Again, this is a direct match so we don't need the range bytes. Finally, the entire packet is sequenced (number 4). The resulting data is:

```
DataGroup Flag     -> 62 (PktData, Tokenized, WantPktSeqd)
Sequence Number    -> 00 01
DataGroup Flag     -> 40 (PktData)
Raw Data Len       -> 82 (compact number for 02)
NBP Func & ID      -> 21 76 (the 2 bytes of raw data)
DataGroup Flag     -> 70 (PktData, Tokenized, LastGroup)
Sequence Number    -> 00 02
```

This time we were able to describe the packet in only 10 bytes. Finally, lets assume that the transmitter needs to resend the above packet with no change. We get the following packet to be processed:

AppleTalk Remote Access Smartbuffering Example:

We will show what happens to an NBP Lookup packet as it is retransmitted and it's id changes. NOTE: AppleTalk Remote Access expects the LAP source and destination to be zeroed. It also expects the length byte to be zeroed on transmission (this is filled in by the other side after being received). We will assume that no packets have yet been transmitted.

```
First Lookup Packet:

LAP/DDP Header      -> 00 00 02 00 00 00 00 11 11 22 22 11 22 02 FE 02
NBP Func & ID       -> 21 74
NBP Tuple           -> 22 22 22 FE 00 01 = 09 AFPServer 01 *
```

Since this is the first packet being sent we do not already have an entry for it or any of it's parts in our transmit cache. Therefore, we will create an entry for the entire packet (sequence number 0) and set the datagroup flag to indicate that the receiver should also create an entry. We also create an entry in our checksum table to the interesting parts of the packet. In this particular case they are the LAP/DDP Header (combined), the NBP Tuple, and of course the entire packet. These checksum entries will give us the possibility to match all or part of a subsequent packet. The resulting packet to be sent is simply a flag followed by the raw data in this case:

```
DataGroup Flag      -> 52 (PktData, LastGroup, WantPktSeqd)
LAP/DDP Header      -> 00 00 02 00 00 00 00 11 11 22 22 11 22 02 FE 02
NBP Func & ID       -> 21 74
NBP Tuple           -> 22 22 22 FE 00 01 = 09 AFPServer 01 *
```

We now have one packet sequenced and 3 checksums that describe it. Now, if the lookup packet moved on to a new ID, we would be presented with the following packet (before Smartbuffering):

```
Second Lookup Packet:

LAP/DDP Header      -> 00 00 02 00 00 00 00 11 11 22 22 11 22 02 FE 02
NBP Func & ID       -> 21 75 (Note the ID change from 74)
NBP Tuple           -> 22 22 22 FE 00 01 = 09 AFPServer 01 *
```

This time, the transmitter checks to see if it already has the entire packet in it's cache and discovers that it does not (remember the ID byte changed). It then checks to see if the LAP/DDP Header matches. It discovers that it does have a match for that part of the packet in sequence number 0, bytes 0 through 0x0f. It outputs the appropriate datagroup flag indicating that a token is being sent for that part of the packet. It also creates a new entry into the cache for this part of the packet (so it will not have to be described as a range in the future) and sets the WantSeqd bit in the datagroup flag (this will be sequence number 1). When it gets to the NBP Func & ID bytes, it simply outputs a datagroup flag that indicates they are raw data. It does not try to checksum or sequence them since they change quite often and it would not save any space to tokenize two bytes. Next, it tries to match the NBP tuple and finds that it has a match for that part of the packet in sequence number 0, bytes 0x12 through 0x24. It also creates a sequence for this portion of the packet and emits the datagroup flag that describes the range found and also indicates to the receiver that it should sequence this part

```
Fourth Lookup Packet:

LAP/DDP Header      -> 00 00 02 00 00 00 00 11 11 22 22 11 22 02 FE 02
NBP Func & ID       -> 21 76 (Note same as above)
NBP Tuple           -> 22 22 22 FE 00 01 = 09 AFPServer 01 *
```

This time the transmitter will check to see if it can match the entire packet and discover that it does have a match in sequence number 4. Therefore, it can describe the packet as being made up of only one sequence, and the following data results:

```
DataGroup Flag      -> 70 (PktData, Tokenized, LastGroup)
Sequence Number     -> 00 04
```

As you can see we were able to describe the original 37 byte packet in only 3 bytes. Typically with NBP traffic, we would reach this stage after only 2 packets because the NBP ID does not normally change with each packet sent.

AppleTalk Remote Access Application Programming Interface (API) External Reference Specifications

Apple Computer, Inc.

AppleTalk® Remote Access Application Programming Interface (API) External Reference Specifications

The AppleTalk Remote Access **API** provides an application programming interface to the Remote Access Manager. It supports calls to load and unload the Remote Access Manager (RAM), create and terminate connections, retrieve the current RAM status, and to determine if a specified network address is local or remote. Optionally, the AppleTalk Remote Access **API** can display a user interface showing the process of a connection. The parameters for the connect call can be a connection document, created with the Remote Access application, or all the parameters of the connection can be specified out right. These specifications also include how to tell if AppleTalk Remote Access is installed, and how to deal with the serial port when AppleTalk Remote Access is in answer mode.

***NOTE:** Although every attempt has been made to verify the accuracy of the information presented, this document may contain errors and is subject to change.*

Gestalts

When fully installed, Remote Access defines several new gestalt selectors. Below are the new defines and explanation of their use.

To see if serial port arbitration is installed, call _Gestalt using these defines.

```
#define gestaltArbitorAttr                          'arb '
#define gestaltSerialArbitrationExists              0
```

For example:

```
OSErr io;
// check to see if serial port arbitrating is installed...
long attribs;
io = Gestalt(gestaltArbitorAttr, &attribs);
if ( ((io == noErr) && (attribs & (1 << gestaltSerialArbitrationExists))) )
   {
   // have serial port arbitration
   }
else
   {
   // no serial port arbitration
   }
```

More information on serial port arbitration is discussed in a section below.

To see if Remote Access Connection Interface is available use these defines:

```
#define gestaltRemoteAccessAttr                     'strm'
#define gestaltRemoteAccessExists                   0
```

To check if Remote Access support is available in the Alias Manager use these defines:

```
#define gestaltAliasMgrAttr                         'alis'      // as defined in GestaltEqu.h
#define gestaltAliasMgrSupportsRemoteAppletalk      1
```

Serial Port Arbitration

When installed Remote Access provides serial port arbitration throught the Serial Port Arbitrator tool. All serial drivers registered with the Communications Resource Manager are arbitrated by the Serial Port Arbitrator.

To check to see if the Serial Port Arbitrator is installed, check the `gestaltSerialArbitrationExists` flag of the `gestaltArbitorAttr` Gestalt selector (see "Gestalts" section for these defines and an example of this call).

If serial port arbitration is present, call OpenDriver when you want to use a serial port. If the driver requested is not open, OpenDriver will return a result of noErr and the reference number of the driver. If the driver requested is open (in use), OpenDriver will return the error `portInUse`. When you are finished using the driver, call CloseDriver. OpenDriver and CloseDriver calls should always be balanced, although the Serial Port Arbitrator will protect against multiple open/close calls.

If serial port arbitration is not present, do not use OpenDriver to determine if the driver is open. It will return you a result of noErr and the reference number, allowing you access to a driver that another application is using. To determine if the serial driver requested is open by another application, you must walk the DCE (Device Control Entry) unit table (see Device Driver chapter of Inside Macintosh vol II).

It is important to use the new method if serial port arbitration is available. Remote Access, when set up for answering calls, is passively using a particular serial driver. If you use the new method, the Serial Port Arbitrator will give up the passive claim and allow your OpenDriver call to return noErr. Later, when you call CloseDriver, Remote Access will again passively claim the port and setup the modem for answering.

Below is a code example in C illustrating how to use serial port drivers under the new and old methods:

```
Boolean HaveSerialPortArbitration()
{
    /*  check to see if serial port arbitrating is installed,
        return true if so.
    */

    OSErr io;
    long attribs;
    io = Gestalt(gestaltArbitorAttr, &attribs);
    return ((io == noErr) && (attribs & (1 << gestaltSerialArbitrationExists)) );
} // HaveSerialPortArbitration
```

```
Boolean CanOpenDriver(unsigned char *driverName)
{
    /*  walks the unit table looking for a match for driverName,
        if found, check to see if the driver is open. return
        false if so.
    */

    Boolean canOpen = false;
    Boolean match = false;
    short index = 0;
    short count;
    DCtlHandle dceHndl;
    unsigned char *namePtr;

    // number of entries in unit table…
    count = *(short*)UnitNtryCnt;

    while ( !match && index < count )
        {
        // get handle to this DCE…
        dceHndl = (DCtlHandle)(*(Handle)((*(Handle)UTableBase) + (index*4) ));

        if ( dceHndl )
            {
            // see if ram based (test bit 6)…
            if ((*dceHndl)->dCtlFlags & 0x40)
                {
                // in ram, so we have a handle…
                namePtr = (*(Handle)((*dceHndl)->dCtlDriver)) + 18;
                }
            else
                {
                // in rom, so we have a pointer…
                namePtr = ((*dceHndl)->dCtlDriver) + 18;
                }

            // compare name: case insensitive, diacritical sensitive…
            if ( RelString((const Str255)driverName,(const Str255)namePtr,false,true) == 0 )
                {
                match = true;
                // see if drvr is open (test bit 5)…
                canOpen != ((*dceHndl)->dCtlFlags & 0x20);
                }
            }
        ++index;   // look at next drvr in unit table
        }

        return canOpen;
} // CanOpenDriver
```

```
void UseSerialPort()
{
    /* illustrate the new way and old way of testing whether
       a serial driver is open or closed.
    */

    short refNum;
    OSErr io;

    if ( HaveSerialPortArbitration() )
        {
        // have serial port arbitration, use new method...

        io = OpenDriver("\p.aout",&refNum);

        if ( io == portInUse )
            {
            // port is in use by another application/etc, we can't use it!
            }
        else
            {
            // use the serial driver

            // close the driver when through...
            io = CloseDriver(refNum);
            }
        }
    else
        {
        // no serial port arbitration, use old method...

        if ( CanOpenDriver("\p.aout") )
            {
            io = OpenDriver("\p.aout",&refNum);

            // use the serial driver

            // close the driver when through...
            io = CloseDriver(refNum);
            }
        }

} // UseSerialPort
```

AppleTalk Remote Access API

Common Parameters

The TRemoteAccessParamBlock is a union of all of the available AppleTalk Remote Access API commands. The TRemoteAccessParmHeader is a struct which consists of a DControlParamHeader followed by a DExtendedParam which is followed by a DRemoteAccessParmHeader. The extendedCode is used to specify the AppleTalk Remote Access API command wanted. The resultStrPtr field returns a Pascal string to indicate what error occurred. If you are not interested in the string, set this field to nil. If you do pass a pointer however, it must point to a buffer of at least 256 bytes in length. If the result of the call happens to be noErr, then the length byte of the string will be zero. Since this version of Remote Access only deals with the user port, the parameter portGlobalsPtr should always be set to zero. The csCode field should normally set to RAM_EXTENDED_CALL and the extendedType is set to REMOTEACCESSNAME. These constants are defined in RemoteAccessInterface.h.

```
#define DControlParamHeader \
      QElem       *qLink;           // next queue entry \
      short       qType;            // queue type \
      short       ioTrap;           // routine trap \
      Ptr         ioCmdAddr;        // routine address \
      ProcPtr     ioCompletion;     // completion routine \
      OSErr       ioResult;         // result code \
      long        userData;         // for use by the user \
      short       unused;           // unused field \
      short       ioRefNum;         // driver reference number \
      short       csCode;           // normally set to RAM_EXTENDED_CALL
                                    // for AppleTalk Remote Access API  calls

#define DExtendedParam \
      DControlParamHeader \
      Ptr         hReserved1; \
      Ptr         hReserved2; \
      Ptr         resultStrPtr; \   // set to zero if result string is unwanted
      Ptr         extendedType;     // pointer to identifier string, normally set to
                                    // REMOTEACCESSNAME for AppleTalk Remote Access API  calls

#define DRemoteAccessParmHeader \
      DExtendedParam \
      short       extendedCode;     // for use by extended call proc \
      Ptr         portGlobalsPtr;   // pointer to globals for this port (0=userport) \

struct TRemoteAccessParmHeader
{
      DRemoteAccessParmHeader
};
typedef struct TRemoteAccessParmHeader TRemoteAccessParmHeader;

union TRemoteAccessParamBlock
{
      TRemoteAccessParmHeader       HDR;          // header pb
      TRemoteAccessParmHeader       LOAD;         // load pb
```

```
    TRemoteAccessParmHeader        UNLOAD;       // unload pb
    TRemoteAccessConnectParam      CONNECT;      // connect pb
    TRemoteAccessDisconnectParam   DISCONNECT;   // disconnect pb
    TRemoteAccessStatusParam       STATUS;       // get current status
    TRemoteAccessIsRemoteParms     ISREMOTE;     // check network address location
    TRemoteAccessPasswordMunger    MUNGEPW;      // run password through munger
    TRemoteAccessGetCodeHooks      CODEHOOKS;    // get internal code hooks
};
typedef union TRemoteAccessParamBlock TRemoteAccessParamBlock;
typedef TRemoteAccessParamBlock *TPRemoteAccessParamBlock;
```

Load

The load command is used ensure that the Remote Access Manager is loaded into memory and must be used before making a Connect call. It uses the standard TRemoteAccessParmHeader as the parameter block. The example code below shows how it works. To use the MungePW command, it is not necessary to use the load command first.

```
#include "RemoteAccessInterface.h"
void LoadRemoteAccess()
{
  TRemoteAccessParmHeader loadPB;

  loadPB.LOAD.csCode = RAM_EXTENDED_CALL;           // extended call
  loadPB.LOAD.resultStrPtr = nil;                   // result string
  loadPB.LOAD.extendedType = REMOTEACCESSNAME;      // to remote access
  loadPB.LOAD.extendedCode = CmdRemoteAccess_Load;  // try to load
  PBRemoteAccess(&loadPB, false);                   // issue sync call
  if (loadPB.LOAD.ioResult)
    ShowError(loadPB.LOAD.ioResult);
}
```

Unload

The unload command is used release the Remote Access Manager and free its memory. It uses the standard TRemoteAccessParmHeader as the parameter block and can be issued immediately after making a Connect call. This allows the Remote Access Manager to be unloaded as soon as an active connection is terminated with a disconnect, if no other clients have loaded Remote Access. Below is an example unload call.

```
#include "RemoteAccessInterface.h"
void UnloadRemoteAccess()
{
  TRemoteAccessParmHeader loadPB;

  // unload the code (will not actually go away till this connection is done)
  loadPB.UNLOAD.csCode = RAM_EXTENDED_CALL;           // extended call
  loadPB.UNLOAD.resultStrPtr = nil;                   // result string
  loadPB.UNLOAD.extendedType = REMOTEACCESSNAME;      // to remote access
  loadPB.UNLOAD.extendedCode = CmdRemoteAccess_Unload; // try to unload
  PBRemoteAccess(&loadPB, false);                     // issue sync call
  if (loadPB.UNLOAD.ioResult)
    ShowError(loadPB.UNLOAD.ioResult);
}
```

Connect

This call is used to initiate an outgoing connection. When you are connected in this mode you will still retain access to your current network. Network numbers are re-mapped in a limited way in order to solve problems of network number conflicts between the two machines being directly connected, thus ensuring they will always be accessible to each other. Unfortunately, it is not possible to solve all of the other possible conflicts due to the limited number of network numbers available. In order to provide a method of ensuring access to all networks on the destination network a guaranteedAccess method is available. When connected in this mode, you will lose access to all services beyond those on the same single network number that the calling machine belongs to. In order to notify clients of the AppleTalk stack that a network will no longer be reachable we have created a new AppleTalk Transition Queue event. (See Network Transition Events later in this document.) When connecting the client passes in a TRAConnectInfoTemplate, or the FSSpec of a document which contains the connect parameters. The connect parameter block contains the optionFlags field which specifies the connect options. The flags are shown below:

```
// connect option flags
#define kNSCanInteract        0x00000001   // User interaction (password prompt) is OK
#define kNSShowStatus         0x00000002   // show the status of the connect or disconnect call
#define kNSConnectDocument    0x00000004   // connect using the specified document using FSSpec
#define kNSPassWordSet        0x00000010   // use the specified password field when connecting
                                           // by document
```

The kNSCanInteract flag allows interaction with the user to get the password if necessary. The kNSShowStatus flag enables the connection status display. The display is modal dialog which updates with new messages as the connection progresses. When the kNSConnectDocument flag is set, the AppleTalk Remote Access API will use the specified document for the connect parameters of the TRAConnectInfoTemplate. The document is specified by the FSSpec record which contains vRefNum (volume reference number), parID (directory ID), and name (pointer to Pascal style string containing the document name). No other parameters need to be supplied. The kNSPassWordSet flag overrides the saved password when connecting by document or by PB and forces AppleTalk Remote Access API to use the passWord field in cleartext. If the kNSPassWordSet flag is clear and the passwordSaved flag is set, then the client must supply a munged password.

In the case that a client is connecting by PB, the fields in the connectInfo record need to be supplied. Within this record is a version which is used to check for compatibility (currently set to 1). The ltType parameter specifies the type of the link tool that will be used in this connection. You specify the length and point to the address used in connecting by setting up addressInfoLength and addressInfoPtr. The ltSpecificTemplatePtr is expected to point to the template of the link tool specific parms. An example of link tool specific parms might be items such as the serial port reference that is used in the MNP Link Tool. A userName is passed in that indicates the name of the user logging in. A passWord is specified if the user is not logging in as a guest. If the user wants to be a guest, the guestLogin flag is set. The connectReminderTimer is used if the caller wants to be reminded that a connection is in progress. This field is set to the number of seconds between reminders, and can be set to zero if no reminders are wanted. If a connectReminderTimer is set you must set the connectOKWaitTimer that indicates how long the reminder dialog will wait for OK to be hit before disconnecting.

```
struct TRAConnectInfoTemplate
{
  unsigned long version;                    // version of this format
  unsigned long ltType;                     // Link Tool type
  long addressInfoLength;                   // length of the address information
  Ptr addressInfoPtr;                       // pointer to connect address info
  long ltSpecificTemplateLength;            // length of the ltspecific information
  Ptr ltSpecificTemplatePtr;                // pointer to link tool specific params
  unsigned char passWord[PASSWORDBUFSIZE];  // user password
  unsigned char userName[USERNAMESIZE];     // user name
  unsigned long connectReminderTimer;       // value for connection reminder in seconds
  unsigned long connectOKWaitTimer;         // how long to wait for OK on reminder timer
  Boolean guestLogin;                       // try to log in as a guest
  Boolean passwordSaved;                    // set if password is saved
  Boolean guaranteedAccess;                 // flag to guarantee access to servers internet
};
typedef struct TRAConnectInfoTemplate TRAConnectInfoTemplate;
typedef TRAConnectInfoTemplate *TPRAConnectInfoTemplate;

struct TRemoteAccessConnectParam
{
  DRemoteAccessParmHeader
  TRAConnectInfoTemplate connectInfo;       // The connection information template
  unsigned long optionFlags;                // bit mapped connect option flags
  FSSpec fileInfo;                          // file info for connect document
};
typedef struct TRemoteAccessConnectParam TRemoteAccessConnectParam;
```

The following is an example connection procedure:

```
#include "RemoteAccessInterface.h"
void DoConnect()
{
  TRemoteAccessConnectParam pb;
  Str255 PathName = "MyHardDisk:Remote Access:Connect Document";

  LoadRemoteAccess();                                   // Get the Remote Access Manager
                                                        // loaded
  pb.CONNECT.csCode = RAM_EXTENDED_CALL;                // extended call
  pb.CONNECT.resultStrPtr = nil;                        // don't want result strings
  pb.CONNECT.extendedType = REMOTEACCESSNAME;           // to Remote Access
  pb.CONNECT.extendedCode = CmdRemoteAccess_DoConnect;  // connect command
  pb.CONNECT.portGlobalsPtr = nil;                      // use the user port
  pb.CONNECT.fileInfo.vRefNum = 0;                      // Use the full pathname
  pb.CONNECT.fileInfo.parID = 0;
  CopyPStr(&PathName,&pb.CONNECT.fileInfo.name);        // copy the string to fileInfo.name

  // Ask for password if needed, use connection document, & show connection status
  pb.CONNECT.optionFlags = kNSCanInteract | kNSConnectDocument | kNSShowStatus;
  PBRemoteAccess(&pb, false);                           // issue sync call
  if (pb.CONNECT.ioResult)
    ShowError(pb.CONNECT.ioResult);                     // Do Error reporting and recovery
```

```
UnloadRemoteAccess();                                   // Unload when disconnected.
}
```

Disconnect

The disconnect command is used to terminate an existing session or cancel one that is being created. If you only want to disconnect a session that was connected with a specific parameter block you can do so by setting a pointer to the parameter block used to issue the connect in abortOnlyThisPB. If you want to disconnect a connection created by anyone, you set the abortOnlyThisPB field to zero. If you are disconnecting an outgoing call, you pass zero in portGlobalsPtr. Disconnecting ports other than the userport, is not supported in this version. You should always set disconnectin to zero. The option kNSShowStatus will cause the AppleTalk Remote Access API to display the status dialog during the disconnect.

```
#define kNumWarnEntriesMax  5                   // number of entries in warn array
struct TRemoteAccessDisconnectParam
{
  DRemoteAccessParmHeader
  unsigned long disconnectin;                   // Note: Set this parameter to 0
  TPRemoteAccessParamBlock abortOnlyThisPB;     // only abort a connection opened by this pb
  unsigned long warnArr[kNumWarnEntriesMax];    // set warn times here in seconds (zero all if
                                                // no warnings)
  unsigned long optionFlags;                    // bit mapped connect option flags
};
typedef struct TRemoteAccessDisconnectParam TRemoteAccessDisconnectParam;
```

The following is an example of a simple disconnect procedure. It will disconnect any existing active connection.

```
#include "RemoteAccessInterface.h"
void DoDisconnect()
{
  TRemoteAccessDisconnectParam pb;

  // set up the Remote Access PB
  pb.DISCONNECT.csCode = RAM_EXTENDED_CALL;                 // extended call
  pb.DISCONNECT.resultStrPtr = nil;                         // don't want result strings
  pb.DISCONNECT.extendedType = REMOTEACCESSNAME;            // to Remote Access
  pb.DISCONNECT.extendedCode = CmdRemoteAccess_Disconnect;  // disconnect command
  pb.DISCONNECT.portGlobalsPtr = nil;                       // user port
  pb.DISCONNECT.abortOnlyThisPB = nil;                      // don't get tied to any specific pb
  pb.DISCONNECT.optionFlags = 0| kNSShowStatus;             // show status while disconnecting
  PBRemoteAccess(&pb, false);                               // issue sync call
  if (pb.DISCONNECT.ioResult)
    ShowError(pb.DISCONNECT.ioResult);                      // Do Error reporting and recovery
}
```

IsRemote

The "IsRemote" command is used to determine if a network address is remote or local. If the network is remote, the call will optionally return the information necessary to make the connection to the remote network. The parameter theAddress contains the network address to be checked. The format of theAddress is the same as for the struct AddrBlock as defined in AppleTalk.h:

Bytes 3 & 2 (High Word): Network Number
Byte 1: Node Number
Byte 0 (Low Byte): Socket Number

The optionFlags parameter is used for getting, or disposing, connection information. The following flags are defined:

```
#define ctlir_getConnectInfo 0x01          // will get connect info if address remote
#define ctlir_disposeConnectInfo 0x02      // will dispose info in connectInfoPtr properly
```

If the ctlir_getConnectInfo flag is set, and the network address is remote, the information necessary to create the remote connection is returned. If the ctlir_disposeConnectInfo flag is set, the connect information structure pointed to by connectInfoPtr, is disposed of properly.

The locationIsRemoteFlag parameter is a flag that is returned true if the network address is remote. The connectInfoLength parameter indicates the length of the connect information. The connectInfoPtr is a pointer to the connection information for connecting with the remote address. These values are returned when the network address is remote, and the ctlir_getConnectInfo flag is set.

```
struct TRemoteAccessIsRemoteParms
{
  DRemoteAccessParmHeader
  long theAddress;                          // address that is to be checked
  unsigned long optionFlags;                // Set to ctlir_getConnectInfo or
                                            // ctlir_disposeConnectInfo, if zero only checks
                                            // theAddress
  Boolean locationIsRemoteFlag;             // returns true if address is remote
  long connectInfoLength;                   // length of the following data
  TPRAConnectInfoTemplate connectInfoPtr;   // The connection information template pointer
};
typedef struct TRemoteAccessIsRemoteParms TRemoteAccessIsRemoteParms;
```

Status

The status command is used to obtain information about Remote Access. The information you can obtain is how long a connection has been active, how much time remains in the connection, the name of the user that made an answering connection, the name of the computer you are connected to on a calling connection, and the last message that was posted. The statusBits parameter is used to determine if a connection is active, starting up, in the process of tearing down, if the connection is an answering or calling connection, if the computer is enabled to receive answer calls or if a disconnect is in progress. The following flags are defined:

```
// bits passed back in statusBits
#define CctlConnected           0x00000001    // set when connected
#define CctlAnswerEnable        0x00000004    // set when we are set to answer calls
#define CctlServerMode          0x00000008    // set for answer mode, clear for call mode
#define CctlConnectionAborting  0x00000010    // connection is being torn down
#define CctlConnectInProg       0x00000020    // set when connection in progress or fully
                                              // connected
#define CctlDisconnectInStarted 0x00008000    // somebody has started a disconnectIn
#define CctlMultiNodeReady      0x80000000    // shows if we currently have a multinode
                                              // address to enable answer mode.
```

The following struct is used when making a status call:

```
struct TRemoteAccessStatusParam
{
  DRemoteAccessParmHeader
  unsigned long statusBits;                 // bits for current status
  unsigned long timeConnected;              // number of seconds we have been connected
  unsigned long  timeLeft;                  // number of seconds remaining in connection
                                            // (0xffffffff infinite)
  unsigned char *userNamePtr;               // returns user name, expects pointer to buffer
                                            // of USERNAMESIZE if non nil
  unsigned char  *connectedToNamePtr;       // returns name of where we connected to,
                                            // expects pointer to buffer of USERNAMESIZE if
                                            // non nil
  TPRemoteAccessParamBlock connectedByParamPtr; // a pointer to the parameter block
                                            // "initiating" the connection if we are
                                            // connected
  TPRemoteAccessParamBlock statusConnectedByParamPtr; // a pointer to the parameter block
                                            // "initiating" the connection when status was
                                            // posted
  unsigned char  *theLastStatusMsgPtr;      // expects pointer to buffer of size
                                            // MAXSTATUSMSGSIZE
  unsigned char  *statusUserNamePtr;        // pointer to buffer of size USERNAMESIZE
  long  statuslttype;                       // link tool type
  long  statusmsgOptionFlags;               // classification of message type
  long  statusMsgNum;                       // specific message number
  long  statusMsgSeqNum;                    // pass in zero if always want status, otherwise
                                            // use last value, if status is new, new number
                                            // is returned
  unsigned long  userSignature;             // signature of port creator
  unsigned long  userRefCon;                // refcon of port creator
};
typedef struct TRemoteAccessStatusParam TRemoteAccessStatusParam;
```

An example status call that determines the state Remote Access is in.

```
#include "RemoteAccessInterface.h"
void GetStatus()
{
  TRemoteAccessStatusParam pb;
```

```
  Str255 UserName,connectedTo,lastMessage;
  long lastSeqNum,statusBits;

  pb.STATUS.csCode = RAM_EXTENDED_CALL;                    // extended call
  pb.STATUS.resultStrPtr = nil;                            // put results here
  pb.STATUS.portGlobalsPtr = nil;                          // do UserPort
  pb.STATUS.extendedType = REMOTEACCESSNAME;               // to Netshare
  pb.STATUS.extendedCode = CmdRemoteAccess_Status;         // status command
  pb.STATUS.userNamePtr = &UserName;
  pb.STATUS.connectedToNamePtr = &connectedTo;
  pb.STATUS.theLastStatusMsgPtr = &lastMessage;
  pb.STATUS.statusUserNamePtr = nil;
  pb.STATUS.statusMsgSeqNum = 0;
  PBRemoteAccess(&pb, false);
  if (pb.STATUS.ioResult)
    ShowError(pb.STATUS.ioResult);                         // Do Error reporting and recovery
  else
  {
    // now decode the flag bits into words
    statusBits = pb.STATUS.statusBits;
    if (statusBits & CctlServerMode)
      printf("Answer connection\n");
    if (statusBits & CctlConnected)
      printf("Calling connection\n");
    if (statusBits & CctlConnectionAborting)
      printf("Cancel in progress\n");
    if (statusBits & CctlAnswerEnable)
      printf("Waiting for incoming call\n");
    if (statusBits & CctlConnectInProg)
      printf("Connection in progress\n");
  }
}
```

MungePW

The MungePW command is used to encrypt a password to be stored in a document. Normally, when connecting by document, it is not necessary to use this command, since the password in a document is stored in encrypted format. It uses a struct TRemoteAccessPasswordMunger with the inputs username pointer and password pointer. The reserved field should always be set to zero. The munged password is return in the data buffer pointed to by passWordPtr. It is not necessary to call the Load command before using MungePW. The maximum username and password lengths are defined in the RemoteAccessInterface.h header file.

```
struct TRemoteAccessPasswordMunger
{
  DRemoteAccessParmHeader
  unsigned char *userNamePtr;                              // pointer to username string
  unsigned char *passWordPtr;                              // user password
  unsigned short reserved;                                 // must set to zero
};
typedef struct TRemoteAccessPasswordMunger TRemoteAccessPasswordMunger;
```

Below is an example routine that calls and gets the password in *passWordPtr munged.

```
#include "RemoteAccessInterface.h"
void MungePassword()
{
  TRemoteAccessPasswordMunger MungePB;
  Str255 UserName = "John Doe";
  Str255 password = "thispass";

  MungePB.MUNGEPW.csCode = RAM_EXTENDED_CALL;          // extended call
  MungePB.MUNGEPW.resultStrPtr = nil;                  // result string
  MungePB.MUNGEPW.extendedType = REMOTEACCESSNAME;     // to remote access
  MungePB.MUNGEPW.extendedCode = CmdRemoteAccess_PassWordMunger;
  MungePB.MUNGEPW.userNamePtr = &UserName;
  MungePB.MUNGEPW.passWordPtr = &password;
  PBRemoteAccess(&MungePB, false);                     // issue sync call
                                                       // and encrypted the eight bytes in
                                                       // password

  if (MungePB.MUNGEPW.ioResult)
    ShowError(MungePB.MUNGEPW.ioResult);
  }
```

GetCodeHooks

The GetCodeHooks command is used to return a pointer to the remapper procedure. This routine can then be called to do special remappings for applications are passing network addresses as part of their data. The call can be made with the clients network number and the node number and this routine will return the remapped equivalents.

```
struct TRemoteAccessGetCodeHooks
{
  DRemoteAccessParmHeader
  RemmaperProcPtr remapperProc;                        // quick vector to remapper code
};
typedef struct TRemoteAccessGetCodeHooks TRemoteAccessGetCodeHooks;
```

The routine returned by this call is defined as follows:

```
pascal void DoRemapper(unsigned long whereNet, unsigned long incomingFlag, unsigned long
sourceSwapFlag, unsigned short *theNet, unsigned char *theNode)
```

whereNet-> This value has the net where this packet just came from or is going to, it is needed to determine if any remapping should even take place.

incomingFlag-> Set to true if data is incoming, false if data is outgoing.

sourceSwapFlag-> Set to true if a source style swap is to be used. A source style swap means that we do remappings based on the address being a source address. If this flag is false, destination style swaps are done.

theNet-> Pointer to unsigned short containing the net to be remapped.

theNode-> Pointer to unsigned char containing the node to be remapped.

Network Transition Events

Network transition events are generated by Remote Access to inform interested clients that network connectivity has changed. The type of change is indicated by the newConnectivity flag. If this flag is true, new connectivity is being added (i.e. a connection to a new internet has taken place). In this case, all network addresses will be returned as reachable. If the newConnectivity flag is false, certain networks are no longer reachable. Since Remote Access is connection based and internally functions much like a router it has knowledge of where a specific network exists. Remote Access can take advantage of that knowledge during a disconnect to inform AppleTalk clients that a network is no longer reachable. This information can be used by the AppleTalk client to age out connections immediately rather than waiting a potentially long period of time before discovering that the other end is no longer reachable.

When Remote Access is disconnecting, it will generate a "Network Transition Event (theEvent=5)" through the AppleTalk transition queue. A client upon receiving such a message can ask Remote Access (through a network validate hook passed to the client) if a specific network is still reachable. If the network is still reachable, true will be returned. A client can then continue to check other networks he is interested in until he has learned the status of each of them. After a client is finished checking his networks he returns to Remote Access where the next AppleTalk transition queue client is called.

Since the "Network Transition Event" is transitional, it is important to realize that the information that the network validate hook returns is only valid if a client has just been called as a result of a transition. In other words, a client can only validate networks when it has been called to handle a "Network Transition Event". It is also important to realize that the "Network Transition Event" can be called as the result of an interrupt, so a client should obey all of the normal conventions involved at being called at this time (i.e. don't ask for memory from the memory manager, etc.).

The following information assumes you have already installed yourself into the AppleTalk transition queue.

ATTransNetworkTransition

The ATTransNetworkTransition event will be generated whenever a network transition occurs. You will be passed the following information using C calling conventions:

```
ClientTransitionHandler(long theEvent, Ptr aqe, TNetworkTransition *thetrans);

theEvent        <---        will be set to ATTransNetworkTransition
aqe             <---        points to transition task struct
thetrans        <---        points to the TNetworkTransition struct
```

The TNetworkTransition struct passed to you is defined as:

```
typedef struct TNetworkTransition
{
  uPtr private;                    // pointer used internally by 976
  ProcPtr netValidProc;            // pointer to the network valid proc
  Boolean newConnectivity;         // true=new connectivity, false=loss of connectivity
} TNetworkTransition;
```

To check a network number for validity the client uses the netValidProc to call Remote Access. This call is defined as follows:

```
long netValidProc(TNetworkTransition *thetrans, unsigned long theAddress);

thetrans        --->        pass in the TNetworkTransition struct given to you when your
                            transition handler was called.
theAddress      --->        this is the network address you want checked.  The format of
                            theAddress is the same as for the struct AddrBlock as defined in
                            AppleTalk.h:
```

Bytes 3 & 2 (High Word): Network Number
Byte 1: Node Number
Byte 0 (Low Byte): Socket Number

Return codes

TRUE	network is still reachable
FALSE	network is no longer reachable

Error Codes

```
// ---------------------------------------------------------------------------
// MNP Error Codes - MNPInterface.h
// ---------------------------------------------------------------------------

#define MNP_ERR_BASE                    -6050             // base for MNP driver errors

#define ERR_MNP_NEGOTIATION_FAILURE     (MNP_ERR_BASE-1)  // Connection parameter negotiation
                                                          // failure
#define  ERR_MNP_CONNECT_TIME_OUT       (MNP_ERR_BASE-2)  // Connect request (acceptor mode)
                                                          // timed out
#define  ERR_MNP_NOT_CONNECTED          (MNP_ERR_BASE-3)  // Not connected
#define  ERR_MNP_ABORTED                (MNP_ERR_BASE-4)  // Request aborted by disconnect
                                                          // request
#define  ERR_MNP_ATTENTION_DISABLED     (MNP_ERR_BASE-5)  // Link attention service is not
                                                          // enabled
#define  ERR_MNP_CONNECT_RETRY_LIMIT    (MNP_ERR_BASE-6)  // Connect (initiator mode) request
                                                          // retry limit reached.
#define  ERR_MNP_COMMAND_IN_PROGRESS    (MNP_ERR_BASE-7)  // Command already in progress.
#define  ERR_MNP_ALREADY_CONNECTED      (MNP_ERR_BASE-8)  // Connection already established.
#define  ERR_MNP_INCOMPATIBLE_PROT_LVL  (MNP_ERR_BASE-9)  // Connection failed due to
                                                          // incompatible protocol levels
#define  ERR_MNP_HANDSHAKE_FAILURE      (MNP_ERR_BASE-10) // Connection handshake failed.
```

```
// ---------------------------------------------------------------------------
// Netshare Error Codes - RemoteAccessInterface.h
// ---------------------------------------------------------------------------

#define ERR_BASE                        -5800

#define ERR_NOTCONNECTED                (ERR_BASE-0)
#define ERR_CONNECTIONABORTED           (ERR_BASE-1)
#define ERR_ALREADYCONNECTED            (ERR_BASE-2)
#define ERR_COMMANDALREADYINPROGRESS    (ERR_BASE-3)
#define ERR_BADVERSION                  (ERR_BASE-4)
#define ERR_INSHUTDOWN                  (ERR_BASE-5)
#define ERR_CONNECTIONABORTING          (ERR_BASE-6)
#define ERR_ALREADYENABLED              (ERR_BASE-7)
#define ERR_ZONEBUFBADSIZE              (ERR_BASE-8)
#define ERR_CONNECTTIMEDOUT             (ERR_BASE-9)
#define ERR_CONNECTUSERTIMEDOUT         (ERR_BASE-10)
#define ERR_BADPARAMETER                (ERR_BASE-11)
#define ERR_NOMULTINODE                 (ERR_BASE-12)
#define ERR_ATALKNOTACTIVE              (ERR_BASE-13)
#define ERR_NOCALLBACKSUPPORT           (ERR_BASE-14)
#define ERR_NOTOPENEDBYTHISPB           (ERR_BASE-15)
#define ERR_NOGLOBALS                   (ERR_BASE-16)
#define ERR_NOSMARTBUFFER               (ERR_BASE-17)
#define ERR_BADATALKVERS                (ERR_BASE-18)
#define ERR_VLD8_CONNECT                0
#define ERR_VLD8_CALLBACK               (ERR_BASE-19)
#define ERR_VLD8_BADVERSION             (ERR_BASE-20)
#define ERR_VLD8_BADUSER                (ERR_BASE-21)
#define ERR_VLD8_BADPASSWORD            (ERR_BASE-22)
#define ERR_VLD8_BADLINK                (ERR_BASE-23)
#define ERR_VLD8_NOCALLBACKALLOWED      (ERR_BASE-24)
#define ERR_VLD8_ALLCBSERVERSBUSY       (ERR_BASE-25)
#define ERR_VLD8_GUESTNOTALLOWED        (ERR_BASE-26)
#define ERR_VLD8_SERVERISIMPOSTER       (ERR_BASE-27)
#define ERR_VLD8_LOGINNOTENABLED        (ERR_BASE-28)
#define ERR_REMOTEPORTALREADYEXISTS     (ERR_BASE-29)
#define ERR_OPENNOTALLOWED              (ERR_BASE-30)
#define ERR_NOUSERSANDGROUPS            (ERR_BASE-31)
#define ERR_PORTSHUTDOWN                (ERR_BASE-32)
#define ERR_PORTDOESNOTEXIST            (ERR_BASE-33)
#define ERR_PWNEEDEDFORENABLE           (ERR_BASE-34)
#define ERR_DAMAGED                     (ERR_BASE-35)
#define ERR_NETCONFIGCHANGED            (ERR_BASE-36)


// ---------------------------------------------------------------------------
// Connection Control Language Error Codes - CCL.h
// ---------------------------------------------------------------------------

#define  cclErr_BaseCode                -6000
#define  cclErr_AbortMatchRead          cclErr_BaseCode          // internal error used to
abort match read
```

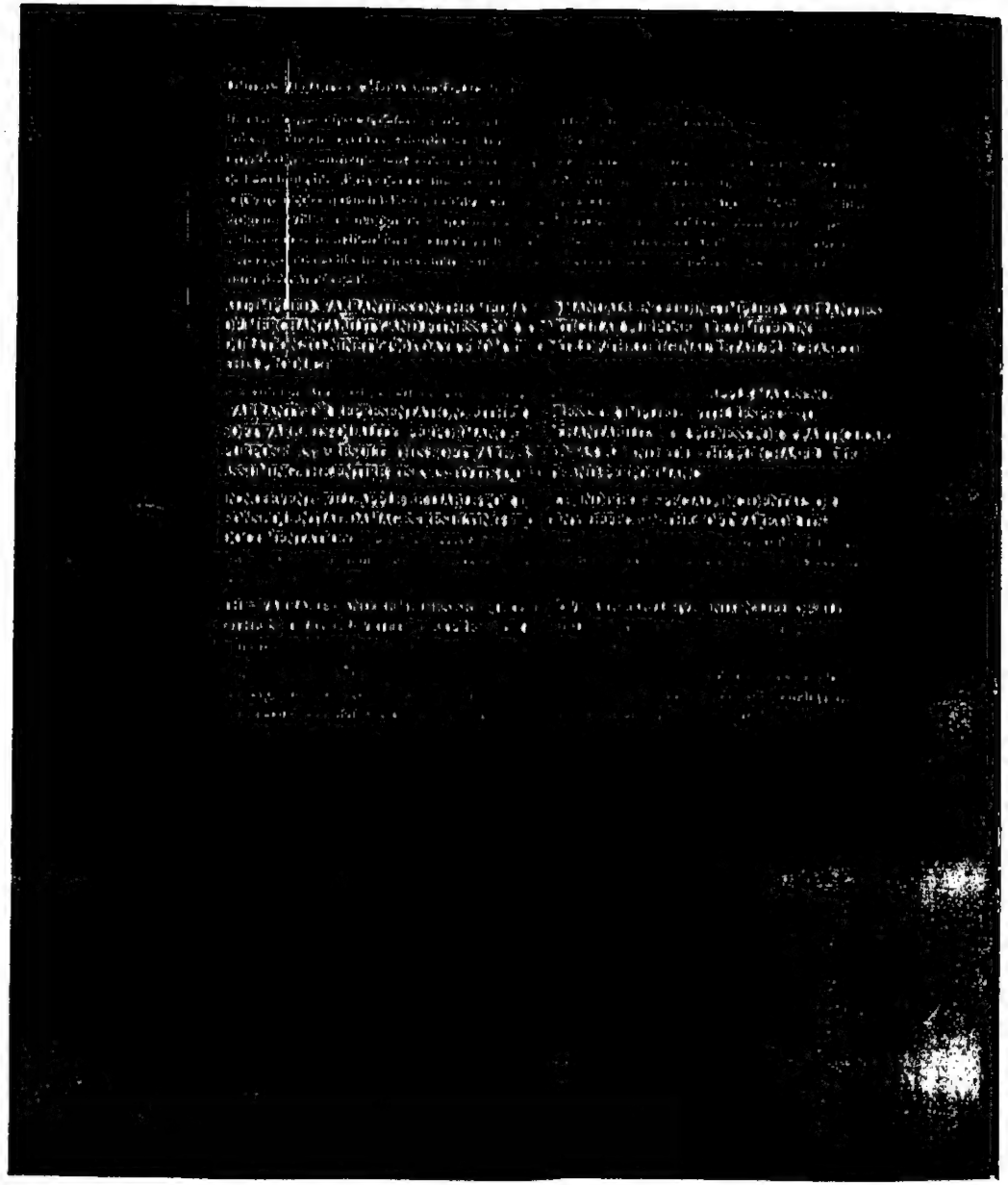
```
#define  cclErr                         (cclErr_BaseCode - 6)     // CCL error base
#define cclErr_CloseError               (cclErr_BaseCode - 7)     // There is at least one
script open
#define cclErr_ScriptCancelled          (cclErr_BaseCode - 8)     // Script Canceled
#define  cclErr_TooManyLines            (cclErr_BaseCode - 9)     // Script contains too many
                                                                  // lines
#define cclErr_ScriptTooBig             (cclErr_BaseCode - 10)    // Script contains too many
                                                                  // characters
#define  cclErr_NotInitialized          (cclErr_BaseCode - 11)    // CCL has not been
                                                                  // initialized
#define  cclErr_CancelInProgress        (cclErr_BaseCode - 12)    // Cancel in progress.
#define  cclErr_PlayInProgress          (cclErr_BaseCode - 13)    // Play command already in
                                                                  // progress.
#define  cclErr_ExitOK                  (cclErr_BaseCode - 14)    // Exit with no error.
#define  cclErr_BadLabel                (cclErr_BaseCode - 15)    // Label out of range.
#define cclErr_BadCommand               (cclErr_BaseCode - 16)    // Bad command.
#define  cclErr_EndOfScriptErr          (cclErr_BaseCode - 17)    // End of script reached,
                                                                  // expecting Exit.
#define  cclErr_MatchStrIndxErr         (cclErr_BaseCode - 18)    // Match string index is out
                                                                  // of bounds.
#define  cclErr_ModemErr                (cclErr_BaseCode - 19)    // Modem error, modem not
                                                                  // responding.
#define  cclErr_NoDialTone              (cclErr_BaseCode - 20)    // No dial tone.
#define  cclErr_NoCarrierErr            (cclErr_BaseCode - 21)    // No carrier.
#define  cclErr_LineBusyErr             (cclErr_BaseCode - 22)    // Line busy.
#define  cclErr_NoAnswerErr             (cclErr_BaseCode - 23)    // No answer.
#define  cclErr_NoOriginateLabel        (cclErr_BaseCode - 24)    // No @ORIGINATE
#define  cclErr_NoAnswerLabel           (cclErr_BaseCode - 25)    // No @ANSWER
#define  cclErr_NoHangUpLabel           (cclErr_BaseCode - 26)    // No @HANGUP

// ---------------------------------------------------------------------------
// Link Tool Manager Error Codes - LTMInterface.h
// ---------------------------------------------------------------------------

#define  ERR_LTM_BASE                   -5900              // base of errors for LTM
#define ERR_LTM_LISTENER_ID_IN_USE      (ERR_LTM_BASE-1)   // Specified  Listener identifier is
                                                           // in use
#define ERR_LTM_NO_LISTENER             (ERR_LTM_BASE-2)   // Listener of specified type is not
                                                           // available
#define ERR_LTM_RESOURCE_NOT_REGISTERED (ERR_LTM_BASE-3)   // Listener of specified type is not
                                                           // available
#define ERR_LTM_PORT_NOT_CLAIMED        (ERR_LTM_BASE-4)   // claim request failed because to
                                                           // port is busy
#define ERR_LTM_COMMAND_NOT_ALLOWED     (ERR_LTM_BASE-5)   // LTM command not allowed on
                                                           // specified port
#define ERR_LTM_BAD_VERSION             (ERR_LTM_BASE-6)   // connect failed due to
                                                           // incompatible LTM versions
#define ERR_LTM_ARBITRATION_TIMEOUT     (ERR_LTM_BASE-7)   // Connection failed due to a time
                                                           // out during the listener
                                                           // arbitration
#define ERR_LTM_KODE_NOT_FOUND          (ERR_LTM_BASE-8)   // kode resource not found in
                                                           // specified file
```

```
#define ERR_LTM_PORT_DISPOSED          (ERR_LTM_BASE-9)   // A Dispose Port call caused the
                                                          // request to fail.
#define ERR_LTM_RESOURCE_CLAIMED       (ERR_LTM_BASE-10)  // call failed because resource is
                                                          // already claimed
#define ERR_LTM_PORT_RESOURCES_CLAIMED (ERR_LTM_BASE-11)  // call failed because the port's
                                                          // resources are already claimed
#define ERR_LTM_RESOURCE_NOT_CLAIMED   (ERR_LTM_BASE-12)  // call failed because the resource
                                                          // was unclaimed
#define ERR_LTM_PORT_RESOURCES_NOT_CLAIMED (ERR_LTM_BASE-13) // The LTM port's resources are
                                                          // NOT claimed.
#define ERR_LTM_PORT_UNCLAIMED         (ERR_LTM_BASE-14)  // LTM listen port unclaimed
#define ERR_LTM_CONNECTION_REFUSED     (ERR_LTM_BASE-15)  // LTM listener refused connect
                                                          // request
#define ERR_LTM_CLAIM_ABORTED          (ERR_LTM_BASE-16)  // LTM claim call aborted due to
                                                          // LTM_ARB_CLAIM_CANCEL call
#define ERR_LTM_END_OF_PORT_LIST       (ERR_LTM_BASE-17)  // End of open port list reached in
                                                          // current port status session

#define ERR_LTM_NOT_CONNECTED          (ERR_LTM_BASE-18)  // Not connected.
#define ERR_LTM_CONNECTION_ABORTED     (ERR_LTM_BASE-19)  // Connection request aborted
#define ERR_LTM_BAD_LENGTH             (ERR_LTM_BASE-20)  // Length of write request exceeds
                                                          // maximum.
#define ERR_LTM_BAD_PARAMETER          (ERR_LTM_BASE-21)  // Bad parameter.
#define ERR_LTM_COMMAND_IN_PROGRESS    (ERR_LTM_BASE-22)  // Command already in progress.
#define ERR_LTM_CONNECTED              (ERR_LTM_BASE-23)  // Connection established.
#define  ERR_LTM_CONNECT_CANCELED      (ERR_LTM_BASE-24)  // Connection request canceled.
#define  ERR_LTM_CONNECT_TIMEDOUT      (ERR_LTM_BASE-25)  // Connection time out.
#define  ERR_LTM_NO_DEFAULTS           (ERR_LTM_BASE-26)  // Could not get default info...
#define  ERR_LTM_GOOD_BYE              (ERR_LTM_BASE-27)  // The driver is going away in a
                                                          // rude fashion
```



AppleTalk Remote Access User's Guide

for Macintosh computers

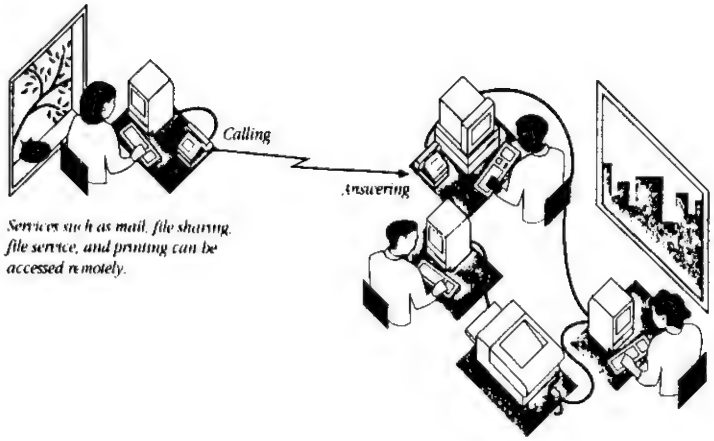
20525 Mariani Avenue
Cupertino, CA 95014-6299
(408) 996-1010

Contents

Introduction

AppleTalk Remote Access is software that lets your Macintosh computer communicate with another Macintosh or an AppleTalk network over standard telephone lines, giving you convenient and direct access to information and resources at a remote location. For example, if you're away from your office but need to get a file from your Macintosh there, Remote Access can make the connection between your home and office computers. When you use Remote Access to call your office computer from your Macintosh at home (or wherever you're using a Macintosh with a modem), you can use Macintosh file sharing to access the files you need.

AppleTalk Remote Access lets you go beyond using just the resources of one computer—if the Macintosh you call is connected to an AppleTalk network, you can take advantage of all the services available on the remote network. For example, you can check your electronic mail, access files on AppleShare file servers, or print to a LaserWriter printer on the network. You can also access host services from your Macintosh, using gateways on the network.



Services such as mail, file sharing, file service, and printing can be accessed remotely.

About this guide

This guide explains how to install and use the AppleTalk Remote Access software. This introductory chapter tells you what you need to get started and provides you with instructions on how to install the software. Chapters 1 and 2 detail how to set up and use Remote Access. Chapter 3 tells you how to keep information on your network secure and how to monitor Remote Access activities. Appendix A provides information on using modems with Remote Access. Appendix B contains troubleshooting suggestions.

This guide assumes you are already familiar with the Macintosh desktop as well as with basic Macintosh skills, such as using the mouse and using the Chooser. If you're unfamiliar with these skills, refer to the manuals that came with your computer.

You should also understand basic networking concepts and have experience using network services like electronic mail, file servers, and printers. For more information on the networking features of your Macintosh, see the networking chapter in the *Macintosh User's Guide* or in your System 7 documentation.

On-screen help

AppleTalk Remote Access includes on-screen information that you can consult when you need help. Balloon Help is a feature of System 7 that explains the function or significance of items you see on the Macintosh screen. To turn on Balloon Help, pull down the Help menu from the Help icon (the question mark) near the right end of the menu bar, and choose Show Balloons. When you point to an item on your screen, a balloon with explanatory text appears next to the item. To turn off Balloon Help, choose Hide Balloons from the Help menu. You can also choose Remote Access Information from the Help menu. Remote Access Information provides you with a series of on-screen cards that explain how to use Remote Access.

For more information

If you want more information about networks and networking, see the following books from Apple Computer (published by Addison-Wesley Publishing Company and available at your local bookstore):

- *Understanding Computer Networks* describes networking basics, including types of networks, network components, network media, and telecommunications.
- *The AppleTalk Network System Overview* provides a technical introduction to the structure and implementation of the AppleTalk network system protocols.
- *Planning and Managing AppleTalk Networks* explains in detail how to administer a small- to medium-sized AppleTalk network.

What you need to get started

To use AppleTalk Remote Access, your Macintosh computer must be running system software version 7.0 or later with at least two megabytes of memory, although four megabytes are recommended. You also need an appropriate Hayes-compatible modem with a data rate of at least 2400 bits per second (bps). See Appendix A for more information on which modems you can use with AppleTalk Remote Access. The Macintosh computer you call must also have AppleTalk Remote Access software installed and a modem attached.

If you received AppleTalk Remote Access in the box with your computer, your package includes:

- one 1.4 MB (high-density) disk, titled *Installer*
- this manual, the *AppleTalk Remote Access User's Guide*

If you purchased AppleTalk Remote Access from an authorized Apple reseller, your package includes:

- two 800K disks, titled *Installer Disk 1* and *Installer Disk 2*
- this manual, the *AppleTalk Remote Access User's Guide*

The *Installer* disk (or disks) include the software and the Installer program that you'll use to install AppleTalk Remote Access software on your Macintosh.

Installing AppleTalk Remote Access on your computer

Follow the steps in this section to install the Remote Access software on your computer.

◆ **Note** The following instructions assume your computer has a floppy disk drive. If you don't have a floppy disk drive, you can install over a network or directly from a desktop Macintosh. See your *Macintosh User's Guide* for more information. ◆

1 **Insert the disk *Installer* (or *Installer Disk 1*) into your floppy disk drive and double-click the Installer icon to open the program.**

The Installer prompts you to identify the software you want to install and the disk where you want to install it.

2 **Select the disk where you want to install Remote Access, then click Install.**

The installation process begins. If you're installing from two 800K disks, the Installer will prompt you to insert *Installer Disk 2* before continuing.

The Installer cannot install Remote Access software on your Macintosh if any application programs are open. You'll receive a prompt if you have any programs open, and you can choose to quit the open programs and continue the installation process or to cancel the installation process.

When the Installer has finished, a dialog box appears to let you know that the installation was successful.

3 **Click Restart to restart your computer.**

Installation is now complete. The Remote Access icon appears on the disk directory, while other components of AppleTalk Remote Access reside in your System Folder. Later, you'll use these other components to set up connection information.

1 Answering Calls With Remote Access

Once you've installed AppleTalk Remote Access software on your Macintosh, you can allow other users to call your computer and, through Macintosh file sharing, you can share and exchange files. If your computer is connected to an AppleTalk network, other users can also access all of the services your network provides, such as file services and electronic mail. Remote users can even use the LaserWriter printers connected to your network. You still have full use of your computer while it answers a call from another user and doesn't need the Remote Access program to be open in order to answer calls—Remote Access will answer automatically in the background. This chapter describes how to set up Remote Access for your Macintosh to answer calls.

Indicating your modem setup

Before your Macintosh can answer calls, you need to connect your computer to an appropriate modem. See Appendix A for a list of modems that you can use with AppleTalk Remote Access, as well as what to do if your modem isn't among those listed. Though you can use a modem capable of transmitting at least 2400 bps, Apple recommends that you use a 9600-bps modem.

The documentation that came with your modem will explain how to set up the modem for use with your computer. Once you've properly connected the modem, you need to register modem information with Remote Access. Follow the steps in this section to indicate your modem setup to the AppleTalk Remote Access software. Make sure you've already installed the Remote Access software according to the instructions in the Introduction.

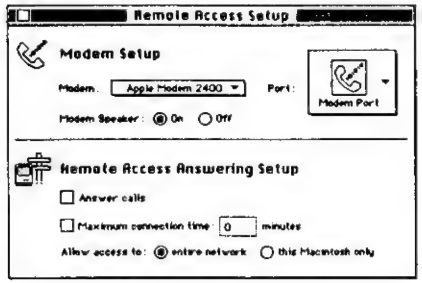
1 **Double-click the Remote Access icon to open the program.**

An untitled *connection document* appears. Ignore this for now, because it's used to call another Macintosh; you'll work with connection documents in Chapter 2.

2 **From the Setup menu, choose Remote Access Setup.**

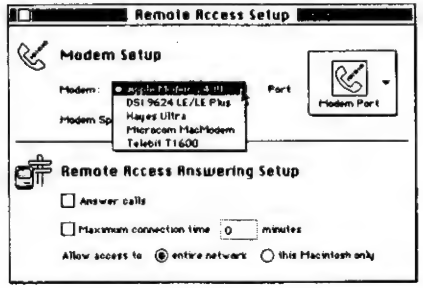
The Remote Access Setup control panel opens. The top portion of the control panel is where you indicate your modem setup.

◆ **Note** Another way to open the Remote Access Setup control panel is to choose Control Panels from the Apple (🍎) menu, then double-click the Remote Access Setup control panel. ◆



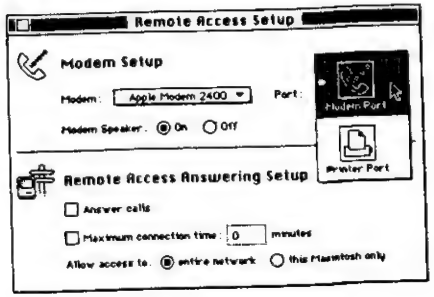
3 **Choose your modem type from the Modem pop-up menu.**

Press the Modem pop-up menu. Drag through the list and choose the name of your modem. If your modem doesn't appear, refer to Appendix A for further information.



4 **Choose the port to which your modem is connected from the Port pop-up menu.**

If you don't know which port your modem is connected to, look at the back of your Macintosh. The port where the modem is connected has an icon above it, which looks like the Modem or Printer port shown in the following figure.



Setting up Remote Access to answer calls

The Remote Access Setup control panel is where you set up Remote Access so your Macintosh can answer calls. Using the lower half of the control panel, you answer calls from users, limit access, and set connection time limits. Follow these steps to set up Remote Access so that other users can call your Macintosh.

1 **If the Remote Access Setup control panel isn't open, choose Remote Access Setup from the Setup menu.**

2 **Click the "Answer calls" checkbox.**

☒ Answer calls

An X should appear in the checkbox to indicate you've selected this option. This lets your Macintosh answer calls from registered users. (Registering individual users is covered later in this chapter.) When this option is selected and your phone rings, your Macintosh answers the call, even if the Remote Access program is not open.

◆ **Note** If you have a service on your phone such as call waiting, then a second incoming call will disconnect you. Contact your local phone company to find out how to turn off call waiting while you use Remote Access. ◆

3 **Click the "Maximum connection time" checkbox, then set the maximum connection time in minutes.**

☒ Maximum connection time: 90 minutes

To ensure that remote connections aren't left idle for a long period of time, you can set your Macintosh to disconnect calls after a certain amount of time. The amount of time depends upon what you expect people to do. Make sure to account for slower transmission rates and to allow users enough time to complete procedures.

When this feature is selected, a user who has called your Macintosh can remain connected for the time specified. The user receives a warning ten minutes before being disconnected, another warning five minutes before, and another warning two minutes before being disconnected. Your Macintosh disconnects the call when the maximum time period, as specified in the box, has passed.

◆ **Note** If you do not select this option, the user can remain connected for an unlimited period of time. However, you can disconnect a user directly (rather than by specifying a time limit), as discussed in the section "Monitoring and Disconnecting a User" later in this chapter. ◆

4 **Make sure you allow access to the entire network.**

Allow access to: ◉ entire network

Do you want callers to have access to all the services on the network to which your computer is connected? If so, the "entire network" button should be highlighted. Otherwise, callers have access only to your Macintosh and the services available on it, such as folders you've shared through file sharing.

5 **Close the Remote Access Setup control panel.**

Preventing users from calling your Macintosh

Follow these steps to prevent all users from calling your Macintosh:

1 **Double-click the Remote Access icon to open the program, if necessary.**

2 **From the Setup menu, choose Remote Access Setup.**

3 **Click the "Answer calls" checkbox.**

☐ Answer calls

Now, no one will be able to call your Macintosh. If you want to prevent access by specific users, you need only revoke their access privileges. This procedure is covered later in this chapter, in the section "Naming a Registered User."

Registering users

By registering users, you let specific users call your Macintosh. You identify each user and give each a password; if you choose, you can also allow guests to call, as explained later in this chapter. Before any user or guest can call, a corresponding user or guest icon must exist in the Users & Groups control panel on your Macintosh.

Important Remote Access uses the same User & Groups control panel as that used for Macintosh file sharing and program linking. If you've already set up a user for file sharing, use the same user icon to set up the user's remote access to your computer.

Naming a registered user

The Remote Access software automatically sets up a user file for you, the Macintosh owner, using the owner name specified in the Sharing Setup control panel for Macintosh file sharing. If you aren't using file sharing, you'll need to set up a user file for yourself according to the instructions in this section. You can register as many as 100 users.

Follow these steps to register a user:

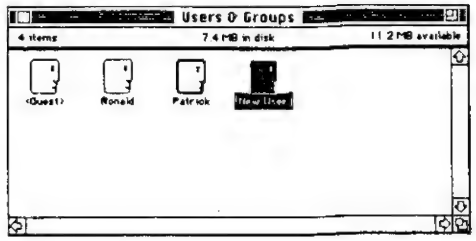
1 **Open the Remote Access program, if necessary.**

2 **Choose Users & Groups from the Setup menu.**

You can also choose Control Panels from the Apple menu, as you normally would to access any control panel, then double-click the Users & Groups icon.

3 **Choose New User from the File menu.**

An icon titled New User appears.



4 **Type the name of the person you want to register.**

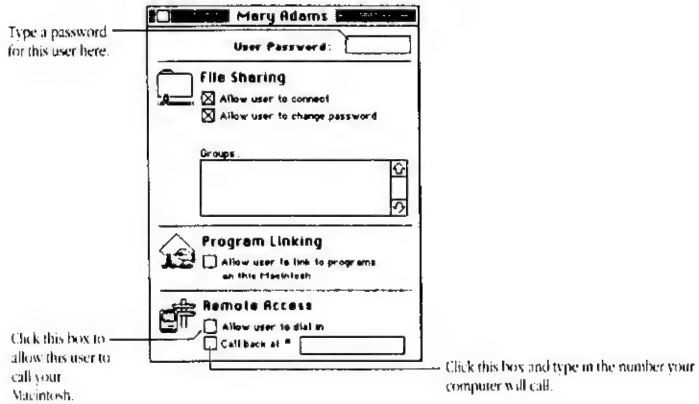
This name must match the spelling the user enters to call this Macintosh, but capitalization doesn't matter. Make sure you notify the user of the exact name you type.

Setting a registered user's password and allowing a user to call

Assigning passwords to registered users is one way of making sure that only authorized callers can gain access to your computer

1 **Open a user icon in the Users & Groups control panel.**

Double-click the user icon you just created. That user's Remote Access options are displayed at the bottom of the window, under the options for file sharing and program linking. For information on file sharing and program linking, refer to the *Macintosh User's Guide* or your System 7 documentation.



2 **Type a password in the User Password text box.**

This password is what the user must enter to call your Macintosh. The password can be up to eight characters long. Make sure to inform the user of the exact password, including capitalization.

3 **Click the "Allow user to dial in" checkbox.**

☒ Allow user to dial in

This lets the user call your Macintosh. Later, if you want to revoke this user's access, you can click the checkbox again to deselect this option.

4 **Click the "Call back at #" checkbox and type in the user's telephone number.**

☒ Call back at #: 9,555-1234

Selecting this option means that when the user calls your Macintosh, your Macintosh will temporarily disconnect the user, then immediately call the user back at the number you've entered. The callback feature is optional; it provides an extra level of security, because the user must call from a predetermined phone number.

The callback number is the telephone number from which the user is calling. You enter the phone number as if you were dialing directly. If your Macintosh needs to dial an outside line (as in many business phone systems), type the outside line number followed by a comma. For example, type `9,5551234`. Type in the area code, if necessary, and enter international phone numbers in the same way. You can type hyphens in the number to make it easier to read, but it is not necessary. Also, if you turn off the callback option, the phone number remains but is unused.

◆ **Note** Remote Access uses touch-tone dialing. For information on pulse dialing and other characters interpreted by the modem, refer to Appendix A. ◆

5 **Close the user window.**

You'll be asked whether to save the information. After you've saved the information, your Macintosh can answer calls from that user.

Allowing guests to call your Macintosh

AppleTalk Remote Access enables you to allow nonregistered users, called *guests*, to call your Macintosh. Unlike registered users, guests do not enter a user name or password when calling your Macintosh.

Important When you allow guests to call your Macintosh, you dramatically reduce its security features. You cannot specify a password or turn on the callback option for guest users. It is strongly recommended that you do not allow guests if your network contains sensitive information. For more information on security, refer to Chapter 3.

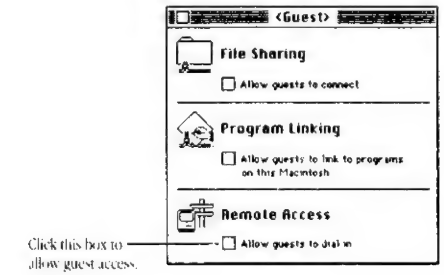
Follow these steps to set up a guest user:

1 **Open the Remote Access program, if necessary.**

2 **Choose Users & Groups from the Setup menu.**

3 **Double-click the <Guest> icon.**

The <Guest> user window opens, which contains information similar to other user windows. The Remote Access options are displayed at the bottom of the window, under the options for file sharing and program linking.



4 **Click the "Allow guests to dial in" checkbox.**

Selecting this option lets guest users call your Macintosh. Later, if you want to prevent guests from calling, click this option again to deselect it.

5 **Close the <Guest> window.**

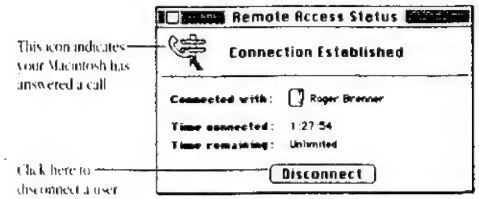
Now the Macintosh will accept calls from any guest.

Monitoring and disconnecting a user

When a user calls your Macintosh, the Remote Access Status window displays the user's name and how long the user has been connected. You also use the Remote Access Status window to disconnect a user.

1 **Choose Status from the Windows menu to view the Remote Access Status window.**

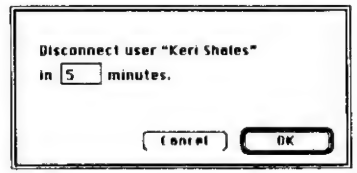
The Remote Access Status window opens.



2 **Click Disconnect to disconnect the user.**

3 **Type the number of minutes that you want to elapse before the user is disconnected.**

Enter 0 to disconnect the user immediately.



4 **Click OK.**

The Remote Access Status window displays the time remaining until the user is disconnected. The user receives a message indicating the amount of time prior to disconnection. The user also receives warning messages 10 minutes, 5 minutes, and 2 minutes before being disconnected, if enough time is allowed.

After you begin the disconnection process, the Remote Access Status window displays a Cancel button. Clicking Cancel halts the disconnection process. The user then receives a message indicating that the disconnection process has been stopped, and the user can remain connected for the amount of time you've specified in the Remote Access Setup control panel.

Quitting the Remote Access program

You can quit the Remote Access program at anytime by using the Quit command from the File menu. If a user is currently connected to your Macintosh, the connection remains open because quitting the Remote Access program has no effect on the connection. Quitting the program frees up memory for your computer to use for other processing.

You'll need to reopen the program if you want to disconnect a user, check the user's status in the Remote Access Status window, or look at the Activity Log (discussed in Chapter 3). If you reopen the program while a connection is active, the Remote Access Status window automatically opens.

2 Calling With Remote Access

You can use AppleTalk Remote Access to call another Macintosh computer that also has Remote Access installed and is set up to answer your calls. If the other Macintosh is attached to an AppleTalk network, you'll have access to the remote network services as if your Macintosh is connected directly. To make calls, you first must install the AppleTalk Remote Access software and supply the necessary connection information.

Setting up Remote Access to make calls

Before you make a call, you need to do the following:

- Install the Remote Access software. Follow the installation procedures in the Introduction.
- Connect a modem to your Macintosh and indicate your modem setup in the Remote Access Setup control panel. Follow the instructions in "Indicating Your Modem Setup" in Chapter 1.
- Make sure that Remote Access software is installed on and a modem is connected to the Macintosh you want to call, and that you can call that Macintosh as a guest or registered user. If you didn't set up the other Macintosh yourself, check with the computer's owner or your network administrator to confirm that the Macintosh is set up to answer calls.

  If the Macintosh you're calling allows registered users only, the other Macintosh must have your name and password in order to answer your call. Verify how your name and password were set by the computer's owner, so that you enter matching information when you call. Also, if the other Macintosh uses callback, check that your telephone number is correct.

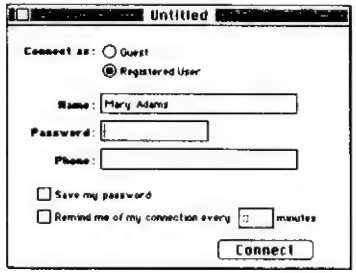
  To answer your call, the Macintosh you're calling does not need the Remote Access program open. However, the other Macintosh and its modem must turned on.
- Obtain the telephone number of the modem connected to the Macintosh you want to call.
- Create a connection document, following the instructions in the next section.

Creating a connection document

The information that AppleTalk Remote Access needs to access another Macintosh computer is stored in a ***connection document***. By using multiple connection documents, you can save the information needed to call different Macintosh computers.

If necessary, double-click the Remote Access icon to open the program.

A new untitled connection document appears on your screen when you open the program. (If you're continuing from the previous chapter, you should have an untitled connection document on your screen.) You can also create a new untitled connection document by using the New command from the File menu.



Connect as a registered user.

You can call another Macintosh as a registered user or as a guest. When you open a new connection document, the Registered User button is selected automatically. Connecting as a guest is covered in the section "Connecting as a Guest" later in this chapter.

3 **Enter your user name in the Name text box, if necessary, and press Tab.**

If the cursor is not in the Name text box, tab to it or click in the box to place the cursor. Your user name must match the spelling that is registered with the Macintosh you want to call, but capitalization doesn't matter.

If you've specified an owner name for file sharing, the owner name appears automatically in the Name text box of the connection document.

4 **Type your password in the Password text box and press Tab.**

Type in your password exactly as registered with the Macintosh you want to call. As you type, the characters appear as bullets (•) to prevent someone else from reading your password. You must match uppercase and lowercase letters exactly.

◆ **Note** If you attempt a connection with an incorrect password, an error message will appear and you can try again. However, as a security precaution, if you try to connect using an invalid password seven times consecutively, the other Macintosh will revoke your ability to make a connection. In this case, you'll need your connection privileges reinstated on the Macintosh you're trying to call. Check with the owner of the other Macintosh or your network administrator if you should need your privileges reinstated; that person must open your user icon in the Users & Groups control panel and reselect the option "Allow user to dial in." ◆

5 **Type the phone number of the Macintosh you want to call.**

Enter the number as if you were dialing directly. For example, if the Macintosh is in area code 818 and its phone number is 555-1234, type:

```
1,818-555-1234
```

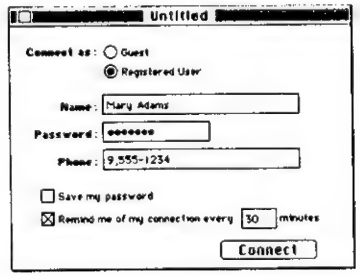
A comma in the phone number designates a pause. You can use dashes for readability, but the program doesn't require them. Also, Remote Access assumes you're dialing from a touch-tone phone. For information on pulse dialing and commands interpreted by the modem, refer to the section "Modem Commands" in Appendix A.

Important The option "Save my password" lets you save your password in the connection document. If you save your password, you won't need to enter it whenever you connect. However, this compromises the security of the computer you're calling, because anyone can then use your connection document to connect. For a discussion of security considerations, including passwords, refer to Chapter 3, "Security and Efficiency Considerations." For now, leave this checkbox blank.

6 **Click the "Remind me of my connection" checkbox if you want Remote Access to remind you periodically that you are connected to another Macintosh. Then type a number between 1 and 9999. This number tells Remote Access how often, in minutes, to remind you of the connection.**

This feature helps you avoid unintentional or prolonged connections. After your computer is connected to another Macintosh for the specified period of time, you will receive a message asking if you want to continue your connection. If you click OK, your work will not be interrupted. If you do not respond within one minute, AppleTalk Remote Access will disconnect you.

At this point, the connection document should look something like the following:



You should now save the connection document, so you can use it later.

From the File menu, choose the Save As command to save the connection document.

Saving the connection document means you won't need to reenter the information when you next call the same Macintosh. You can use the Save As or Save command, or click the connection document's close box. You'll then receive a prompt to enter a name for the connection document. Pick a name that describes the Macintosh you're calling, like "Boston Office" or "Mac at home." The connection document icon will appear in the folder where it is saved. You can store connection documents wherever you like on your hard disk.

Mac at home

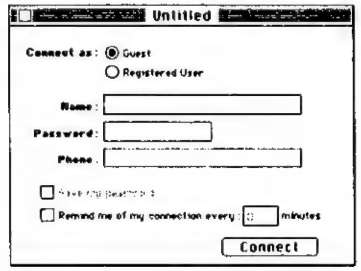
Changing or deleting connection information

After you've saved a connection document, double-click its icon to open it. You can edit any of the information previously entered and save it for later use.

To get rid of a connection document, drag it to the Trash and use the Empty Trash command in the Finder to delete it.

Connecting as a guest

To connect as a guest, open a connection document and click the Guest button.



The text boxes for Name and Password are unavailable, because you don't use those to connect as a guest. You can save the connection document as you would any connection document, so at some later time you can connect using the same information. Note that to connect as a guest, the Macintosh you're calling must allow guest access. Check with the owner of the Macintosh you're calling to find out if you can connect as a guest.

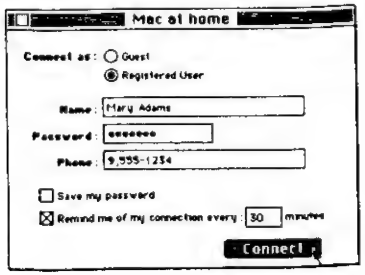
Connecting to a remote Macintosh or network

Once you've installed Remote Access and have created a connection document, you can make a call to another Macintosh.

1 **Double-click a connection document icon.**

This opens the connection document and the Remote Access program.

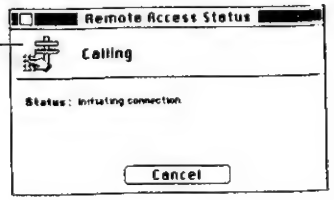
2 **Enter your password, if necessary.**



3 **Click Connect.**

When you click Connect, AppleTalk Remote Access calls the other Macintosh and tries to connect to it. The Remote Access Status window appears to allow you to cancel the connection attempt before the connection is completed and to show the progress of your connection.

This icon indicates that you've called another Macintosh

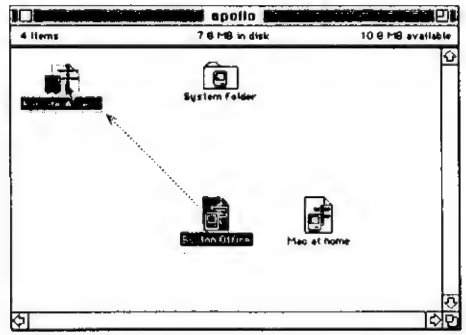


The other Macintosh verifies your name and password, and the connection is then established.

◆ **Note** If the other Macintosh has callback turned on, you are temporarily disconnected from the other Macintosh while it attempts to call you back. Your phone will ring and the modem will answer the call. Then, the connection is established. Refer to Chapter 3 for more information on callback. ◆

Other ways to connect

Another way you can open the Remote Access program is to drag the connection document onto the Remote Access icon.



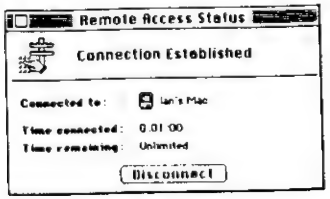
This opens the Remote Access program and the connection document. You then click the Connect button in the connection document window to call another Macintosh.

As a shortcut, hold down the ⌘ key when double-clicking the connection document or when dragging the connection document onto the Remote Access icon. The program will then automatically try to connect using that connection document.

Viewing your connection status

Whenever the Remote Access program is open, you can view your connection status by looking at the Remote Access Status window. To open the window, choose the Status command from the Windows menu.

The Remote Access Status window appears, showing details of your connection.



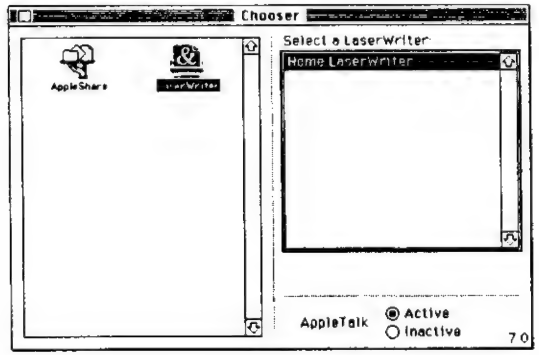
The Remote Access Status window tells you where you're connected and for how long you've been connected. Also shown is the time remaining for your connection. The time limit is set by the owner of the Macintosh you've called, or you may have no time limit for your connection, indicated by the word "Unlimited." You can click the close box to close the Remote Access Status window.

The Windows menu also contains the Activity Log command. The Activity Log contains a listing of all events involving Remote Access, such as connections made or attempted. Using the Activity Log is covered in Chapter 3.

Selecting remote network services

AppleTalk Remote Access allows you simultaneous use of the services of your local AppleTalk network as well as those of the network of the Macintosh you've called, provided the other Macintosh is set up to allow access to the remote network. Once you've established a connection to a remote AppleTalk network with Remote Access, you select network services in the Chooser window or use other network products such as electronic mail, following the same steps you would to use the service on your local network.

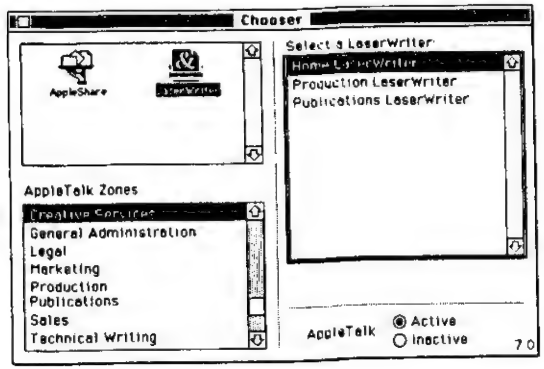
As an example, the following steps show how to print to a LaserWriter that's on a remote AppleTalk network. Let's say you have not yet used Remote Access to connect. Assuming your Macintosh has an AppleTalk network printer such as a LaserWriter attached, the Chooser window shows you the local services available to you:



Now, call the other Macintosh, establishing a connection to the remote AppleTalk network.

1 **Look at the Chooser window.**

The Chooser window displays the network services and AppleTalk zones that are currently available to you. Notice the LaserWriters available from the remote network:



2 **Select a LaserWriter as you normally would, then close the Chooser.**

You can now print to the remote LaserWriter.

Disconnecting from the remote network

Once you establish a connection with a remote network, the Remote Access Status window displays a Disconnect button.

1 **From the Windows menu, choose Status.**

2 **Click Disconnect.**

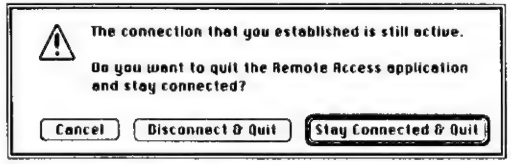
When you click Disconnect, the connection between your computer and the other Macintosh is broken. However, the Remote Access program is still open, so you can make another connection or create a new connection document.

If for some reason you shut down your Macintosh while connected to a remote network, the connection is automatically broken.

Quitting the Remote Access program

You can quit the Remote Access program at any time by using the Quit command from the File menu. If you're currently connected to a remote network, your connection remains established because quitting the program has no effect on the connection. Quitting the program frees up memory for your computer to use for other processing. You cannot quit if you're in the process of connecting or disconnecting.

If you quit while you're connected to a remote network, you'll see a dialog box that reminds you of your connection. You must choose to disconnect and quit, stay connected and quit, or cancel.



Remember that you'll need to reopen the program if you want to disconnect, open a connection document, or check your connection status in the Remote Access Status window. If you reopen the program while a connection is active, the Remote Access Status window automatically opens.

3 Security and Efficiency Considerations

To get the most out of AppleTalk Remote Access, you can take certain steps to make sure your Macintosh is secure and that you're using Remote Access efficiently. Remote Access provides several measures of security, including the ability to monitor users. This chapter discusses efficiency measures and Remote Access security along with general considerations for keeping your network secure.

Using the Activity Log

The Activity Log documents all Remote Access events. It keeps track of when a Remote Access connection is attempted, established, or disconnected, indicating if the connection is a connection you initiated or if another user called your Macintosh. The Activity Log retains the 1,000 most recent activities.

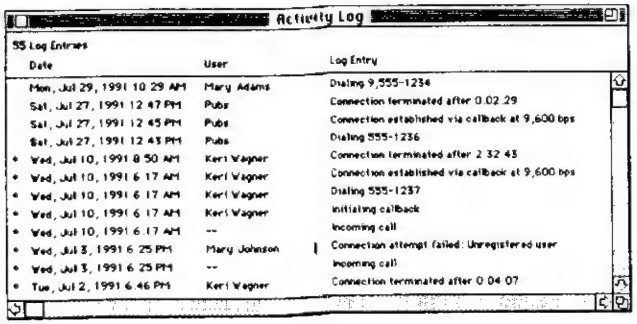
Follow these steps to view the Activity Log:

1. **Double-click the Remote Access icon to open the program, if necessary.**

2. **From the Windows menu, choose Activity Log.**

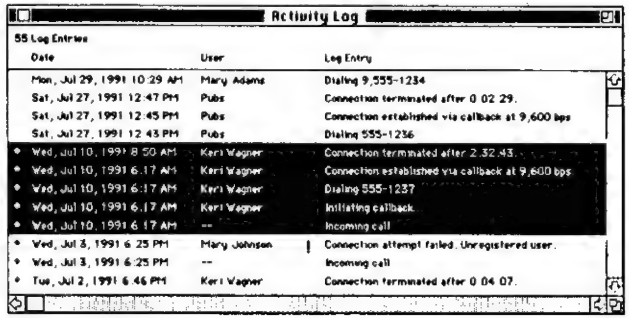
The Activity Log appears displaying the current and most recent activity of your Macintosh. The first column shows the date and time of the activity. A diamond (◆) preceding the first column indicates that your Macintosh established the connection by answering a call from another user. The second column displays the user name.

The third column describes the activity. An exclamation point (!) precedes the activity if an unusual event occurred. This alerts you to activities such as an unknown user trying to call or a user trying to connect with an incorrect password.



Copying, saving, and clearing the Activity Log

To save the Activity Log for your records, you can select the Activity Log entries and copy them to the Clipboard. Click the entry or Shift-click the range of entries you want to copy, or use Select All from the Edit menu, then use the Copy command from the Edit menu.



The entries are copied to the Clipboard as tab-delimited text, so you can paste this into a spreadsheet file. Use this spreadsheet file to save multiple records of the Activity Log, to sort and search through the information, or to print.

You can also save the Activity Log as a text file. To do this, choose Save As from the File menu. You'll be prompted to enter a name for the text file, which is saved on the disk you specify. By default, the text file is saved under the name Log, followed by a range of dates that correspond to the earliest and latest activities in the Activity Log. Use any word-processing or spreadsheet program to view or edit the text file.

Should you wish to clear the Activity Log completely, choose Clear Log from the Edit menu. A dialog box will appear, prompting you to confirm that you wish to clear the Activity Log. After you clear the Activity Log, those entries cannot be recovered, so you should save the log before clearing it.

Maintaining security for your Macintosh and network

Before allowing users to call your Macintosh, make sure you've established what information you want others to have access to. You should set up file sharing for those folders you want other people to use. If you do not use file sharing, a user can still call your Macintosh but will not have access to your computer's folders or hard disks.

If you want to call your own Macintosh, then turn on file sharing but do not share any folders. This way you, as the owner, always have access to all your files under file sharing.

If you are setting up Remote Access on a Macintosh that is part of an AppleTalk network, security is an especially important issue. Remote Access includes several features that provide security for your network. You can register users, use the callback option, and limit the access users have to the network.

Register users

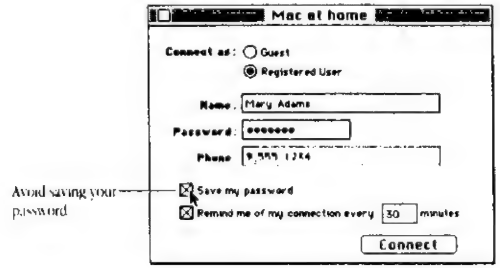
When you call another Macintosh, you gain access to all public files, file servers, and printers on the network to which that Macintosh is connected (but only if the remote Macintosh is set up to allow access to the entire network). Each Macintosh using Remote Access stores the name and password of every user registered to call. Registered users are assigned passwords that they must enter each time they call the Macintosh. A user's name and password act as a key—no one can enter the network without knowing and entering both of them correctly. You should encourage users to change their passwords frequently.

Don't allow guest access

By allowing guests to call your Macintosh, you allow unknown users to access your network. If network security is a concern, then do not allow guest users to call.

Avoid saving your password

Avoid saving your password with your connection document (the "Save my password" checkbox should not be checked). If the password is saved, anyone can use that connection document to call the Macintosh that you are registered to use.

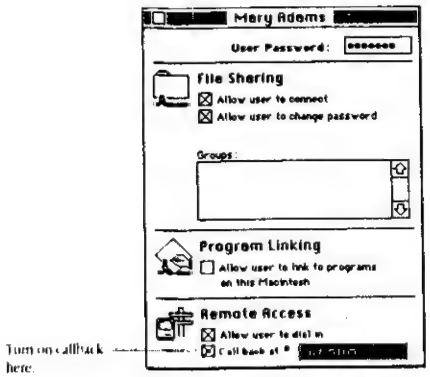


Limits to connection attempts

When a user calls a Macintosh, the user is allowed seven consecutive tries to enter the correct password. After seven incorrect password entries, the user's connection privileges are revoked and the Macintosh owner must set up the user again to call that computer. This seven-try limit helps prevent an unregistered user from guessing a password.

Select callback

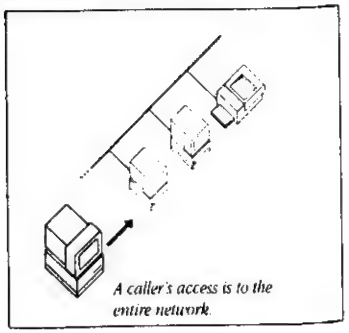
After a user calls and is verified as a registered user, the Macintosh can temporarily break the connection and, using a phone number stored with the user's name, call the user back. This callback feature gives further assurance that the user is authentic, because the user must call from a predetermined phone number. You can set the callback option for each user in the user's information window in the Users & Groups control panel.



Deny access to the network

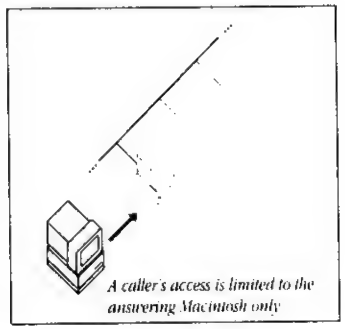
You can limit users' access to only the Macintosh called, preventing all users from accessing services on the remote network.

Allow access to entire network



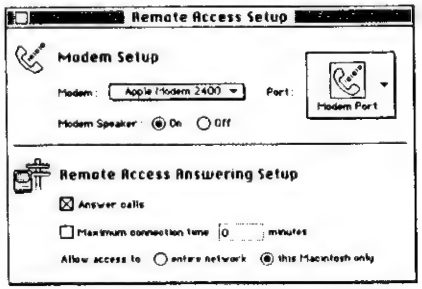
A caller's access is to the entire network.

Allow access to this Macintosh only



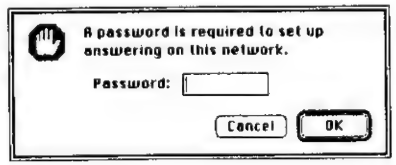
A caller's access is limited to the answering Macintosh only.

To limit access, open the Remote Access Setup window. Click the button to allow access to "this Macintosh only."



Require a password to set up answering on the network

On a large network, the network administrator can require users to enter a password to set up answering on the network. Check with your administrator to see if this feature is implemented and to find out your network password. If the feature is implemented, you'll receive a dialog box when you set up your Macintosh to answer calls in the Remote Access Setup control panel.



Using Remote Access efficiently

The following sections discuss how to use Remote Access efficiently and address special concerns for hardware setup.

Connecting efficiently

Remember that you are using standard telephone lines to connect to a remote network. Just as you are billed for your toll calls and long-distance calls, you are also billed if you call a Macintosh outside your local calling area.

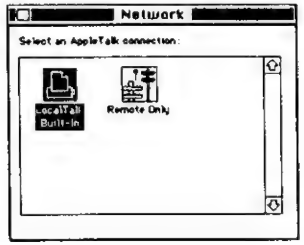
If a Macintosh you're calling has the callback option turned on, the other Macintosh disconnects your call and immediately calls you back. The other Macintosh then originates the phone connection and becomes responsible for the charges.

The following guidelines can help keep your long-distance phone charges to a minimum:

- Disconnect as soon as you no longer need to use the remote network.
- Do not open programs such as word processors, spreadsheets, and graphics programs on the remote network. Because the program will run over telephone lines, commands may take an extraordinarily long time to complete.
- Prepare files as much as possible before you make a connection.
- Set a maximum connection time to limit users who call your computer.
- Use the option "Remind me of my connection." This keeps you from forgetting about a connection, because you must respond to the prompt or Remote Access disconnects you.
- As soon as you finish using a file server volume, drag the volume to the Trash to disconnect from the server.
- While you're using file server volumes, don't leave their windows open. These windows are updated periodically, causing unnecessary network traffic.

Selecting remote network services only

When AppleTalk Remote Access is installed, the Network control panel contains an additional icon called Remote Only. You use the Network control panel to select the type of network to which you want to connect locally. The default is LocalTalk, which uses the printer port on your Macintosh to connect to AppleTalk networks. You may also see other icons depending on your network setup.

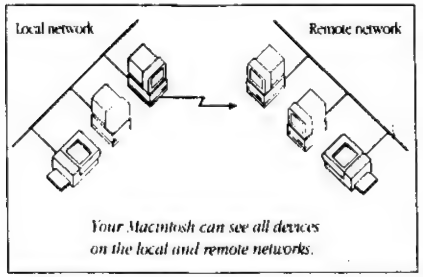


You might select Remote Only for two reasons:

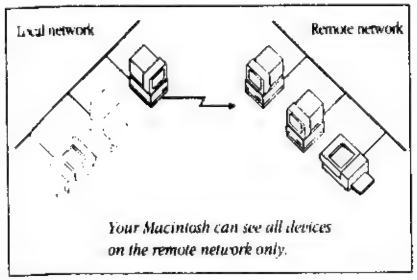
1. To turn off access to your local AppleTalk network (LocalTalk, EtherTalk, or TokenTalk), so you see only the remote network when you make a Remote Access connection.

   Sometimes you can't locate a service on the remote network because a service on your local network has the same address. This is similar to phone numbers, for which two people in different area codes may have the same seven-digit phone number. If you don't specify the area code and dial only the seven-digit number, you call the person who is in the same area code as you. Selecting Remote Only is like dialing the area code, meaning you call the person outside your local area. When you select Remote Only, access to the local network is shut off, ensuring that all services on the remote network are accessible.

With LocalTalk, EtherTalk, or TokenTalk selected



With Remote Only selected



2. To turn off LocalTalk to free up the printer port on your Macintosh for other uses, such as printing to an ImageWriter or StyleWriter.

   If you want to use your printer port for something other than LocalTalk, maybe to print to a StyleWriter or ImageWriter, you need to select Remote Only. This shuts off access to the local network, leaving the printer port free for another use. Since Remote Access is a type of AppleTalk connection, making AppleTalk inactive in the Chooser means you can't use Remote Access. So if you want to use Remote Access and the printer port, you need to select Remote Only.

Be aware that if your Macintosh is currently connected to a remote network and you switch from one AppleTalk connection type to another in the Network control panel, such as from LocalTalk or EtherTalk to Remote Only, you will be disconnected. However, you can always reconnect.

Appendix A: Using Modems With AppleTalk Remote Access

AppleTalk Remote Access includes scripts to use many popular Hayes-compatible modems within the United States. This appendix lists the modems that have scripts included with the AppleTalk Remote Access software and discusses what you can do if your modem is not included. This appendix also shows how to connect a typical modem to your Macintosh and lists standard modem commands.

Modems you can use with Remote Access

AppleTalk Remote Access includes scripts for the following modems for use within the United States. If you're using AppleTalk Remote Access outside of the U.S., contact your authorized Apple reseller or representative for modem information. For the best performance, Apple suggests you use a 9600-bps modem.

2400 bps/V.22bis modems

- Apple Modem 2400 (supports all 2400-bps Apple modems)
- Abaton InterFax 24/96
- Global Village Teleport
- Hayes SmartModem 2400
- Microcom Microporte 1024
- Practical Peripherals 2400SA
- Prometheus ProModem 2400

- Supra SupraModem 2400
- US Robotics Courier 2400e

9600 bps/V.32 and 14400 bps/V.32bis modems

- DSI 9624 LE/LE Plus
- Hayes Ultra 96
- Farallon Remote V.32
- Microcom MacModem V.32
- MultiTech MultiModemV32
- Practical Peripherals 9600SA
- Prometheus ProModem Plus/Ultima
- Telebit T1600
- US Robotics Courier V.32bis

For every type of modem that Remote Access supports, a script resides in the Extensions folder within your System Folder. Modem scripts contain program code used by the Remote Access software to communicate with that particular modem type.

If your modem is not included

Your modem should conform to the CCITT (Comité Consultatif International Télégraphique et Téléphonique) standards V.22bis, V.32, or V.32bis. If you want to use Remote Access with a modem not listed above, a script may be available for your modem. Check with one of the following sources to see if a modem script is available:

- your modem vendor
- your authorized Apple reseller or representative
- the Apple Customer Assistance Center (1-800-776-2333 in the U.S.)
- if you're a subscriber, the Apple Technical Coordinator Answerline or the Apple Software Development Answerline

Some modems not included with the Remote Access software may need a cable that supports CTS (Clear To Send). Call the Apple Customer Assistance Center for more information.

Writing modem scripts

If you understand connection control languages (CCLs) and are an experienced programmer, you may be able to write the necessary modem script yourself. The AppleTalk Remote Access Modem Toolkit is available through APDA (Apple Programmers and Developers Association). The kit includes a HyperCard stack and appropriate documentation to help you build scripts.

APDA offers convenient worldwide access to over 300 Apple and third-party development tools, resources, and information for anyone interested in developing applications on Apple platforms. To order products or get additional information contact:

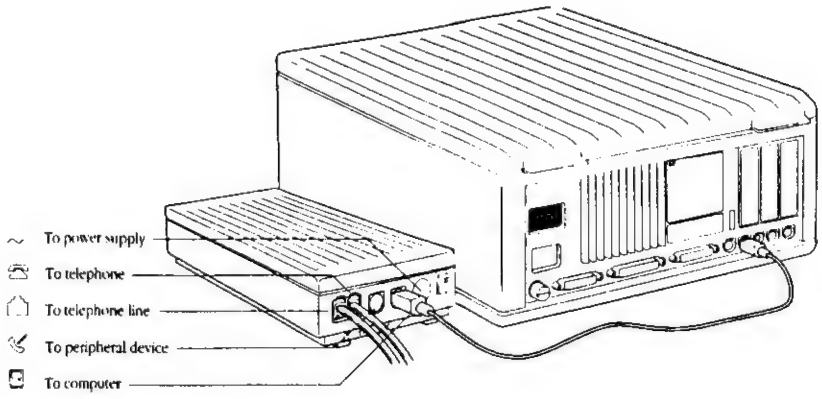
APDA
Apple Computer, Inc.
20525 Mariani Avenue, M/S 33-G
Cupertino, California 95014-6299
800-282-2732 (United States)
800-637-0029 (Canada)
408-562-3910 (International)
408-562-3971 (Fax)
171-576 (TELEX)
AppleLink: APDA

Connecting your modem

How you set up your modem depends on the type of modem you're using. The documentation that came with your modem should provide specific details on how to connect your modem to your computer. Depending on the type of modem or Macintosh you use, you'll need to do the following:

- Connect your modem to your telephone line.
- Connect your modem to your telephone (optional).
- Connect your modem to your Macintosh.
- Connect a power supply to your modem.

The following picture offers a typical example that shows an Apple Data Modem connected to a Macintosh:



Modem commands

Standard commands are interpreted by most modems available in the United States. You enter these commands as characters in the telephone number sent to the modem. The following table lists standard commands, but you should refer to your modem manual for a complete description of the commands it interprets.

Command (character)	*Function*
0–9, #, *	Digits and characters used for dialing.
T	Causes all digits following to be touch-tone dialed (the default).
P	Causes all digits following to be pulse dialed.
,	Pauses before continuing.

Pulse dialing

AppleTalk Remote Access assumes you're dialing from a touch-tone phone. If you require pulse dialing, precede the number with a P, as shown here:

```
P1,818-555-1234
```

If you need a combination of pulse and touch-tone, precede the numbers with P (for pulse) and T (for touch-tone) as appropriate. You may need to do this if you're making a long distance call with pulse dialing, and you first connect to your long-distance carrier's lines then enter an access code using touch-tone. In this case, the phone number you enter should look something like this:

```
P1,818-555-1234 T342
```

Appendix B: Troubleshooting

While using AppleTalk Remote Access, you may occasionally encounter a problem. This appendix provides information that is common to most modems and offers suggestions for solving problems.

You can't make a call.

- Make sure your modem is connected properly and turned on. Refer to your modem documentation if you have any problems.
- Make sure your modem type and port are selected in the Remote Access Setup control panel.

Your modem is working, but you can't connect to another Macintosh.

- Make sure you're a registered user on the other Macintosh. Check that you've entered your user name and password correctly.
- Find out if the other Macintosh is set up to call you back, and verify that your telephone number is correct.
- Make sure the other Macintosh and its modem are turned on, and the Macintosh is set up to answer calls.

You are losing the connection.

- You may have noisy phone lines. Contact your phone company.
- Make sure you've selected the correct modem type in the Remote Access Setup control panel.
- The call-waiting feature can disrupt the connection when someone tries to call you. When you enter the phone number in the connection document, precede the number with the code to turn off call waiting. However, if the other Macintosh has callback turned on, your modem first hangs up, which reenables call waiting. Contact your phone company for information on how to turn off call waiting.

The response time from the remote network is slow.

- Make sure the Macintosh you've called is using a 9600-bps modem. Look in the Activity Log to check the speed the two modems are using.
- Refer to the section "Using Remote Access Efficiently" in Chapter 3 to see if you can save time during the connection.

After you connect to a remote network, you have problems with your ImageWriter or another device connected to the printer port.

- Select Remote Only in the Network control panel to redirect AppleTalk and allow you to print to your serial printer. Refer to the section "Selecting Remote Network Services Only" in Chapter 3 for more information.

After you connect to a remote network, you cannot access your local network.

- Select LocalTalk (or EtherTalk or TokenTalk) in the Network control panel to access your local network. Refer to the section "Selecting Remote Network Services Only" in Chapter 3 for more information.

Other programs that use the serial port don't function properly, you see a dialog box that says the serial port is in use, or your Macintosh freezes during shutdown.

- When the option "Answer calls" is selected in the Remote Access Setup control panel, the serial port is in use by Remote Access. To use other programs that need the serial port, turn off the option "Answer calls."
- If your other program is incompatible with Remote Access, contact your authorized Apple Reseller for more information.

Index

REMOTE-ACCESS (MODEMS)

The following article was written by Jim Kateley and originally posted to the Tech Info Library. Due to many requests for sample scripts, we have decided to also post the article here within Apple SW Updates.

AppleTalk Remote Access: Sample V.32/Slower Script

Article Created: 15 June 1992

TOPIC ----------

I have a new V.32 modem that I want to use with AppleTalk Remote Access (ARA), but there's no script on the application disk that I can use with it.

DISCUSSION ----------

The following AppleTalk Remote Access script for the Telebit QBlazer modem is a good example of how to write a script for any given V.32 (or slower) modem.

Notes

- This discussion assumes that you have access to the documentation for AppleTalk Remote Access and that you are familiar with the command syntax and scripting basics.

- The command syntax is in the AppleTalk Remote Access Modem Scripting Language Guide. You can order the guide from APDA at 800-282-2732 (USA), 800-637-0029 (Canada), or 408-562-3910 (International).

- If you have access to AppleLink, look for related AppleTalk Remote Access script files in the Software Sampler folder.

- For more information, use "AppleTalk Remote Access and V.32bis" as a search string.

```
! "Telebit QBlazer Modem - 12/10/91" JFK
! Adapted from the 7/29/91 T1600 script that ships with ARA 1.0
! 12/10/91 JFK - The QBlazer script is only one line different from the T1600,
!                 but in my grand style of "Give them so much more they choke,
!                 and then you can run around them..." I added the
!                 "Ton'o'comments".
!                 And fixed the one line, of course...
! 12/11/91 JFK - Figured out that S51 needs to be set to 252 on the QBlazer
!                 instead of 255 like on the T1600 and T3000.
! 3/30/92  JFK - Added comments about the hang up sequence.
! 4/3/92   JFK - Added sbreak in hangup label, added S38 for ATH behavior,
!                 moved some commands around for consistency.
!
@ORIGINATE
@ANSWER
!
! Talk to the modem at 9600 bps.  The QBlazer should auto-baud this
```

```
! unless the user has locked the port to a particular speed.  If it
! is locked to a different speed, the user will need to change that.
!
serreset 9600, 0, 8, 1
!
!  The idea here is to get the modem into a known state, and then change only
!  the registers that are necessary to support the connection.  Most of the
!  time AT&F will be sufficient, but some modems allow the user to change the
!  F0 parameters.  There isn't much that can be done to prevent this, but if
!  the modem has any pre-configured configurations that will set most of the
!  required parameters, use it.
!
! Recall the factory configuration
!
! AT&F0 set:
! S61=1 - Go into command mode when receiving break from DTE (see
!         @HANGUP for why the script cares about this).
!
!  Every time the script needs to send commands to the modem, the strategy is:
!  Clear all matchstrings, look for specific responses, and loop around a
!  couple of times.  Later in the script, certain loops pause 50-70 seconds,
!  such as when the script dials a number and is waiting for a connection.
!  Other times, the script pauses 3-5 seconds and loops around. When the script
!  is sending commands to the modem, it should expect to see a response within
!  a couple of seconds, so it's best to look quickly and exit with an error in
!  a reasonable amount of time so the users do not wait a for a long time
!  before they are notified that they may need to power-cycle/reset the modem.
!  When the script is dialing out over a telephone system or PBX, it needs
!  enough time to make a connection.  In short, if it's communicating to a
!  modem, loop in 3-7 second increments.  If the script is waiting for
!  something other than a modem response (like a completed connection or
!  terminal server) it may need 60-70 seconds.
!
!  If the defaults cannot be set, jump down to label 59, which exits and asks
!  the user to check out the modem.  If an  AT&F command will not be accepted,
!  the modem may be hung and needs to be manually reset.
!
settries 0
matchclr
@LABEL 1
matchstr 1 3 "OK\13\10"
write "AT&F0\13"
matchread 20
inctries
iftries 2 59
! Modem is not responding, reset and send a break
DTRClear
pause 5
DTRSet
SBreak
jump 1
!
!  The script was able to get the modem into a default factory state.  Now
!  set the basic hardware type configuration such as command echo, hardware
!  handshaking, and DTR control.  If the &F9 command had not set up handshaking
```

```
!  this is where it would be done.  It's not desirable to create one long
!  command string with everything on it because some modems cannot handle a
!  long command string, and long strings are harder to debug.  It's easy
!  to enter an incorrect S-register value.  For the most part, the following
!  commands are probably common across a lot of modems, but always look up the
!  commands in the modem manual.  For V.32 or slower modems, there should not
!  be any kind of flow control between the modem and the Macintosh.  This is a
!  different situation from V.32bis and faster modems (which require hardware
!  handshaking).
!
! Next, Set up the configuration: drop connection after losing DTR
! Turn off auto answer, command echo, and no DTE flow control.
!
! &D3    - DTR on/off resets modem
! S0=0   - Don't answer calls
! E0     - Turn command echo off
! S58=0  - No DTE flow control
!
@LABEL 3
matchclr
matchstr 1 4 "OK\13\10"
write "AT&D3S0=0E0S58=0\13"
matchread 30
jump 59
!
!  Now that the modem hardware & flow control parameters are set, make sure any
!  protocol negotiation is disabled, and issue any modem specific features
!  here.  Make sure that MNP4/V.42, and MNP5-10/V.42bis negotiations are
!  disabled.  By the way, some V.32/V.32bis modems have an option to disable
!  Trellis error control, which is part of the physical layer modulation.
!  This is not the same as MNP/V.42, and you do not want to disable it!
!
!  Make sure that the modem is configured so it does NOT require error control
!  to complete a link.  ARA 1.0 does all error correction/data compression in
!  software.  All ARA wants is the fastest raw data pipe it can get.  If the
!  script spends time trying to negotiate some error control, the modems and/or
!  Remote Access may time out.
!
!  Also note the S38 configuration.  It is noted later in the script that it is
!  desirable to ensure that the modem's buffer has transmitted all of its data
!  before it actually hangs the modem up.  This ability appears to be
!  implemented on a lot of modems.
!
!  This set of commands is going to be implemented differently on different
!  vendors modems.  In this example, Telebit uses S registers. Other modems may
!  use S registers (but different registers), or \ commands, or % commands; you
!  get the idea.  (Did I mention that you really, really want to have your
!  modem manual handy?)
!
!  It is important that the modem is configured so that it returns
!  the connected speed, NOT the DTE speed.  The script needs to know what the
!  real line speed is in order to set ARA's internal timers.  Some
!  modems don't have the option to display the line speed.  In that case the
!  performance of the connection may not be optimal.
!
```

```
! Next, disable MNP/error control, internal buffering, delay before
! disconnect, and issue extended result codes
!
! S180=0 - Turn off all error detection/correction (ARA does MNP and
!          compression itself.  It needs these turned off in the modem).
! S181=0 - Turn off DTE <-> line buffering if there is no error control.
!                           The idea is to have the Macintosh communicate
!                           with the modem at the line speed of the modem.
! S38=255  - Wait until the modem's buffer is clear OR the other modem
!            disconnects after an ATH is issued before dropping the line.
!            This is done to ensure that all/any data in the modem's buffer
!            has been transmitted to the remote modem before it disconnects.
!            If the remote connection does not receive the
!            disconnect packet (usually the last one sent) it could take
!            up to 45 seconds for the remote connection to timeout and
!            disconnect.
! X2     - Issue extended result codes.  This will display busy, connect XXX,
!          etc.  X2 will say "CONNECT XXX"  Where XXX is the line speed.  This
!          is so ARA can determine what speed the modems are communicating at
!          to set the serial port speed.
!
@LABEL 4
matchclr
matchstr 1 5 "OK\13\10"
write "ATS180=0S181=0S38=255X2\13"
matchread 30
jump 59
!
!  The modem should now be properly configured.  Now check to see if the user
!  has turned off the modem speaker.  If they have, send an additional command
!  to turn it off.
!
! If speaker on flag is true, jump to label 8.  Otherwise turn off the speaker.
!
@LABEL 5
ifstr 2 8 "1"
matchstr 1 8 "OK\13\10"
write "ATM0\13"
matchread 30
jump 59
!
! The modem is ready so enable answering, or originate a call.
!
@LABEL 8
ifANSWER 30
note "Dialing ^1" 3
write "ATS0=0DT^1\13"
!
!  Be aware that different modems will have different format strings
!  to return connection results.  You need to understand the different possible
!  strings and set this area (and then answer area at label 31) to the
!  appropriate value.  Also, remember that the modem was configured to return
!  the connect speed if possible (The X2 command up at label 5).  It's also
!  useful if the modem can return busy, no dialtone, etc. since the script will
!  be able to exit quicker and let the user know what is going on.
```

```
!
!  Also note that the script waits at the bottom of label 9 for 70 seconds,
!  rather than looping around.  Why?  Well, if the script re-issues the dial
!  command too soon, that would cause the modem to hang up.  At this point the
!  script should wait a reasonable amount of time for one of these strings to
!  return from the modem and take the appropriate action.
!
@LABEL 9
matchstr  1 11 "CONNECT 1200\13\10"
matchstr  2 12 "CONNECT 2400\13\10"
matchstr  3 13 "CONNECT 4800\13\10"
matchstr  4 14 "CONNECT 9600\13\10"
matchstr  5 15 "CONNECT FAST\13\10"
matchstr  6 50 "NO CARRIER\13\10"
matchstr  7 50 "ERROR\13\10"
matchstr  8 52 "NO DIALTONE\13\10"
matchstr  9 53 "BUSY\13\10"
matchstr 10 54 "NO ANSWER\13\10"
matchread 700
jump 59
!
!  All that is done for different connect speeds is to set the serial port
!  speed on the Macintosh to match the line speed.
!
@LABEL 11
note "Communicating at 1200 bps." 2
serreset 1200, 0, 8, 1
jump 16
!
@LABEL 12
note "Communicating at 2400 bps." 2
serreset 2400, 0, 8, 1
jump 16
!
@LABEL 13
note "Communicating at 4800 bps." 2
serreset 4800, 0, 8, 1
jump 16
!
@LABEL 14
note "Communicating at 9600 bps." 2
serreset 9600, 0, 8, 1
jump 16
!
@LABEL 15
note "Communicating at 19.2 kbps." 2
serreset 19200, 0, 8, 1
!
!  At this point the modems have connected.  If the script is answering a
!  telephone call, just exit right away and starting communicating.  If the
!  script is dialing out, give the other end some time (3 seconds in this
!  example) to get ready to talk to this modem.  Exit 0 tells Remote Access
!  that the script was successful in attempting a connection.
!
@LABEL 16
```

```
ifANSWER 17
pause 30
@LABEL 17
exit 0
!
!  Notice that the @ANSWER label is actually a comment here, and that
!  @ORIGINATE and @ANSWER start at the same place.  What's the point of having
!  separate entry points if they are not used?  Well, in the case of modems,
!  when they dial out or wait for a call, the setup is usually the same.  One
!  reason for separate entry points is when the script is not directly talking
!  to a modem, but maybe to a PBX or terminal server.  It may be necessary to
!  have completely different configuration for answering and originating
!  connections.
!
! @ANSWER
! Set up the modem to answer
!
@LABEL 30
write "ATS0=1\13"
matchstr 1 31 "OK\13\10"
matchread 30
jump 59
!
!  What is userhook 1 doing in label 32?  Here's the idea:  Either this script
!  controls a server that is waiting to answer the telephone, or it's waiting
!  for a callback to a connection that was initiated.  AppleTalk Remote Access
!  does a "passive" listen on the serial port (via the Serial Port Arbitrator)
!  so that other communications applications can use the serial port when ARA
!  is not using it.  When a call comes in for a server or callback, there
!  will be about 5-14 seconds while the modems negotiate the connection.
!  What would happen if a communications application on this Macintosh
!  wanted to use the serial port during that time?  Both connections
!  would fail.  The userhook 1 command tells ARA to mark the serial port in
!  use.  When that happens, applications that want to use the serial port will
!  be told it's busy, and the incoming connection can complete.  With that in
!  mind, the strategy below is:  When the modem receives a ring, jump to label
!  32, issue the userhook 1 command, then jump back up to label 31, wait for
!  the connect result code and continue processing the script.
!
@LABEL 31
matchstr 1  32 "RING\13\10"
matchstr 2  11 "CONNECT 1200\13\10"
matchstr 3  12 "CONNECT 2400\13\10"
matchstr 4  13 "CONNECT 4800\13\10"
matchstr 5  14 "CONNECT 9600\13\10"
matchstr 6  15 "CONNECT FAST\13\10"
matchstr 7  50 "NO CARRIER\13\10"
matchstr 8  50 "ERROR\13\10"
matchstr 9  52 "NO DIALTONE\13\10"
matchstr 10 53 "BUSY\13\10"
matchstr 11 54 "NO ANSWER\13\10"
matchread 700
jump 31
!
@LABEL 32
```

```
userhook 1
note "Answering phone..." 2
jump 31
!
!  These are some common error messages when the line is busy, no dialtone,
!  etc. They are documented in the Scripting Language Guide.  When the script
!  exits with a code other than zero, Remote Access knows that the connection
!  failed, and will inform the user with a dialog.
!
! 50: error messages
!
@LABEL 50
exit -6021
!
@LABEL 52
exit -6020
!
@LABEL 53
exit -6022
!
@LABEL 54
exit -6023
!
@LABEL 59
exit -6019
!
! Hang up the modem
! Note:  Why try to enter command mode and hang up the line with ATH, when
!        de-asserting DTR will always work, and it is used as a last resort
!        anyway?  If DTR is used immediately,  the modem will hang up
!        immediately.  This can have the ill effect of hanging up before all
!        the data in the modem's internal transmit buffer has been sent.
!        It is very desirable to have the last byte of data sent make
!        it out of the modem and across the phone line.  Typically,
!        the last packet sent is the disconnect packet, and if
!        the other side misses this packet, it may have to wait up to 45
!        seconds to hang up.
!
@HANGUP
@LABEL 60
settries 0
@LABEL 61
!
!  Here's the basic logic for hanging up: If the modem can be configured
!  to enter command mode when it receives a short break, send a short
!  break.  Send an ATH to hang the line up (and if possible up in the
!  configuration, set the modem to attempt to send all the data in the
!  buffer before it disconnects).  If that fails, it must still be on
!  line, so send the escape sequence to try to drop into command mode.
!  Don't issue a short break again since it did not work the first time.
!  If that fails, de-assert DTR which should force the modem to hang up
!  (make sure the cable is wired properly for this option!).
!  If +++ worked, don't send a short break again; flush the serial port
!  buffer in case the ATH failed due to any stray data hanging around.
!
```

```
!  How was this sequence determined?  Trial and error.  Different vendors'
!  modems behave differently when disconnecting.  Some modems will not enter
!  command modem during a disconnect, and the only option is to de-assert DTR
!  to force them to reset.  That's why DTR resets the modem instead of just
!  disconnecting it!  Experiment with this sequence to make it function, but it
!  should work with the majority of the modems available.
!
! Now, since the Telebit modems will drop into command mode when they receive
! a short break (S61=1), issue one here.  This will speed up the disconnect
! sequence by about 5-6 seconds.  Then continue on with normal AT disconnect
! processing.
!
Sbreak
!
! Wait just a brief amount of time (1/2 second in this case) so the modem will
! be ready to accept the ATH command.  Pause 1 actually seems to work ok, but
! it's set to 5 just to be safe.
!
pause 5
write "ATH\13"
matchclr
matchstr 1 63 "OK\13\10"
matchstr 2 63 "NO CARRIER\13\10"
matchstr 3 63 "ERROR\13\10"
matchread 30
inctries
iftries 3 63
! no response, try escape sequence
write "+++"
matchclr
matchstr 1 62 "OK\13\10"
matchread 15
!
! No response from modem, toggle DTR
!
DTRClear
pause 5
DTRSet
jump 61
! Pause 1 second to ensure we meet the escape time delay
@LABEL 62
pause 10
Flush
write "ATH\13"
matchstr 1 63 "OK\13\10"
matchstr 2 63 "NO CARRIER\13\10"
matchstr 3 63 "ERROR\13\10"
matchread 30
jump 61
!
!  Now that the modems have disconnected, and the script has possibly reset,the
!  modem, restore the factory settings.  Remember, the script may have hung up
!  the modem in order to get ready for a callback, or it wants to get ready to
!  wait to answer a call again.
!
```

```
! Recall factory settings
!
@LABEL 63
matchclr
matchstr 1 64 "OK\13\10"
write "AT&F0\13"
matchread 30
!
!  Now turn off auto answer if it was turned on to answer a call.  If this
!  script controls a server, the @ANSWER sequence will be called by ARA.
!  One other thing to watch out for here is that some modems expect to
!  talk to the DTE at the last connected speed.  If this is a V.32
!  modem and it just finished a connection with a 2400 baud modem, it
!  doesn't necessarily want to talk at 2400 the next time!  Some modems
!  don't exhibit this behavior, so play with it and see what happens.  Finally,
!  since it successfully hung up, exit the script with a result code of 0 to
!  let Remote Access know everything worked.
!
! Turn off auto answer, set S51 so modem will check interface speed on
! next command.  This is different than the Telebit T1600/T3000, which wants
! S51=255.
!
! S51=252 - Automatic speed selection, type ahead not permitted.
! S0=0    - Don't answer the phone if it rings.
!
@LABEL 64
matchclr
matchstr 1 65 "OK\13\10"
write "ATS51=252S0=0\13"
matchread 30
!
@LABEL 65
exit 0
```

REMOTE-ACCESS (MODEMS)

The following article was written by Jim Kateley and originally posted to the Tech Info Library. Due to many requests for sample scripts, we have decided to also post the article here within Apple SW Updates.

AppleTalk Remote Access: Sample V.32bis Script

TOPIC----------

I have this new V.32bis modem, and I'd just love to use it with AppleTalk Remote Access. There isn't a script on the application disk that I can use with it. What should I do?

DISCUSSION----------

This discussion assumes that you have access to the documentation for AppleTalk Remote Access, and are familiar with the command syntax and scripting basics.

The following AppleTalk Remote Access script for the Telebit T3000 modem is a good example of how to write a script for any given V.32bis modem.

There is additional information available on AppleLink that discusses specific issues surrounding using ARA with V.32bis modems.

The command syntax can be found in the AppleTalk Remote Access Modem Scripting Language Guide, available from APDA. You can reach APDA for telephone orders at 800 282-2732 USA, 800 637-0029 Canada, and 408 562-3910 International.

```
! "Telebit T3000 Modem 11/11/91" JFK
! 11/14/91 JFK - Added a ton o' comments...
! 11/23/91 JFK - Cleaned up comments a little bit more,
!                 and added S11=50.
! 12/2/91  JFK - Took S11 back out.
! 12/9/91  JFK - Added the pinouts for the RTS/CTS cable to the comments
!                 Since you have to make a hardware handshaking cable.
! 1/16/92  JFK - Add a bunch more comments about the cable.
! 3/30/92  JFK - Added comments about the hang up sequence.
! 3/31/92  JFK - Added sbreak in hangup sequence to speed things up.
! 3/31/92  JFK - Added S38=3 to help ensure that the modem will be
!                 able to transmit all the data in its buffer (including
!                 the disconnect command for the remote modem!) before
!                 it hangs up.
! 4/2/92   JFK - Changed S38 to 255.  See notes below.
! 4/2/92   JFK - Changed S51 to 254 in hang up sequence since 19200 is
!                 our DTE speed.
!
!  Note the cable requirements when using a V.32bis modem.  Since a lot
!  of people do not have this cable, you should not use this approach when
!  scripting for V.32 or slower modems.
!
! Cable needed to use AppleTalk Remote Access with V.32bis modems:
!   Din-8    DB-25
! 1 (DTR)    4,20 (RTS, DTR)
! 2 (CTS)    5 (CTS) *
! 3 (TxD-)   2 (TD)
```

```
! 4 (SG)     7 (SG)
! 5 (RxD-)   3 (RD)
! 6 (TxD+)   Not Connected
! 7 (GPi)    8 (DCD)
! 8 (RxD+)   7 (SG)
!
! * Normally 2(CTS)<-!6 (DSR) on other Macintosh cables.
!
!- One consequence of using this cable is that DSR (or DCD) from the modem is
!  no longer connected to the Macintosh.  This does not let your Macintosh
!  communication software use the DSR (or DCD) signal to detect carrier loss.
!  And since the Macintosh Serial driver does not support the GPi input...you
!  are sort of stuck, unless your communications software does use the GPi
!  input.  Or Apple builds GPi support into the serial driver.
!
!- Since DTR and RTS are connected together, the modem must be configured
!  to ignore DTR (usually the &D0 command) when using this cable with other
!  communications applications.  Otherwise, when RTS handshaking from the Mac
!  is used, the modem will drop connection the first time the Mac de-asserts
!  RTS.
!
!- If there is a need to use DTR to make the modem disconnect, RTS handshaking
!  cannot be used to control the flow of data from the modem to the Macintosh
!  CTS handshaking (from the modem to the Macintosh) is available.  This is
!  what ARA does so it can force the modem to hang up, and at the same time
!  the modem can signal the Macintosh to stop sending data.  This assumes that
!  the Macintosh will always be able to accept data from the modem.  This will
!  not be true if the Macintosh is talking to the modem at 57.6KBps with
!  V.32bis & V.42Bis.  There will be times when the Macintosh will need to
!  signal the modem to stop sending data.
!
!In summary, with this cable:
!
!If you want to use RTS hardware handshaking, you cannot use DTR to control the
!modem.  You will have to resort to other methods to coerce the modem to
!disconnect.
!
!If you want to control the modem with DTR, you cannot use RTS hardware
!handshaking so the Macintosh must be able to accept data from the modem at all
!times, or can recover if data is lost.
!
!In either case you can use CTS hardware handshaking so the modem can signal
!the Macintosh to discontinue sending data.
!
@ORIGINATE
@ANSWER
!
! Talk to the modem at 19,200 bps.  the T3000 should auto-baud this
! unless the user has locked the port to a particular speed.  If it
! is locked to a different speed, the user will need to change that.
!
serreset 19200, 0, 8, 1
!
!  The idea here is to get the modem into a known state, and then change only
!  the registers that are necessary to support the connection.  Most of the
```

```
!  time AT&F will be sufficient, but some modems allow the user to change the
!  F0 parameters.  There isn't much that can be done to prevent this, but if
!  the modem has any pre-configured configurations, and one of them sets
!  hardware handshaking, use it.
!
! Recall the factory configuration
! F9 is the built in pre-configured setting for CTS/RTS handshaking on the
! T3000.  Since it's possible for the user to modify F0 parameters, this is a
! little safer.
!
! AT&F9 sets:
! &C1 - DCD is on after connect
! &D2 - DTR on/off disconnects
! S58=2 - Use RTS/CTS flow control in full-duplex mode
! S61=1 - Go into command mode when receiving break from DTE (see
!         @HANGUP for why the script cares about this).
!
!  Every time the script needs to send commands to the modem, the strategy is:
!  Clear all matchstrings, look for specific responses, and loop around a
!  couple of times.  Later in the script, certain loops pause 50-70 seconds,
!  such as when the script dials a number and is waiting for a connection.
!  Other times, the script pauses 3-5 seconds and loops around. When the script
!  is sending commands to the modem, it should expect to see a response within
!  a couple of seconds, so it's best to look quickly and exit with an error in
!  a reasonable amount of time so the user does not wait a for a long time
!  before they are notified that they may need to power-cycle/reset the modem.
!  When the script is dialing out over a telephone system or PBX, it needs
!  enough time to make a connection.  In short, if it's communicating to a
!  modem, loop in 3-7 second increments.  If the script is waiting for
!  something other than a modem response (like a completed connection or
!  terminal server) it may need 60-70 seconds.
!
!  If the defaults cannot be set, jump down to label 59, which exits and asks
!  the user to check out the modem.  If an  AT&F command will not be accepted,
!  the modem may be hung and needs to be manually reset.
!
settries 0
matchclr
@LABEL 1
matchstr 1 4 "OK\13\10"
write "AT&F9\13"
matchread 30
inctries
iftries 2 59
! Modem is not responding, reset and send a break
DTRClear
pause 5
DTRSet
SBreak
jump 1
!
!  The script was able to get the modem into a default factory state.  Now
!  set the basic hardware type configuration such as command echo, hardware
!  handshaking, and DTR control.  If the &F9 command had not set up handshaking
!  this is where it would be done.  It's not desirable to create one long
```

```
!  command string with everything on it because some modems cannot handle a
!  long command string, and long strings are harder to debug.  It's easy
!  to enter an incorrect S-register value.  For the most part, the following
!  commands are probably common across a lot of modems, but always look up the
!  commands in the modem manual.
!
! Next, Set up the configuration: drop connection after losing DTR
! Turn off auto answer and command echo.
!
! &D3 - DTR on/off resets modem
! S0=0 - Don't answer calls
! E0 - Turn command echo off
!
@LABEL 4
matchclr
pause 5
matchstr 1 5 "OK\13\10"
write "AT&D3S0=0E0\13"
matchread 30
jump 59
!
!  Now that the modem hardware & flow control parameters are set, make sure any
!  protocol negotiation is disabled, and issue any modem specific features
!  here.  Make sure that MNP4/V.42, and MNP5-10/V.42bis negotiations are
!  disabled.  By the way, some V.32/V.32bis modems have an option to disable
!  Trellis error control,  which is part of the physical layer modulation.
!  This is not the same as MNP/V.42, and you do not want to disable it!
!
!  Make sure that the modem is configured so it does NOT require error control
!  to complete a link.  ARA 1.0 does all error correction/data compression in
!  software.  All ARA wants is the fastest raw data pipe it can get.  If the
!  script spends time trying to negotiate some error control, the modems and/or
!  Remote Access may time out.
!
!  Also note the S38 configuration.  It is noted later in the script that it is
!  desirable to ensure that the modem's buffer has transmitted all of it's data
!  before it actually hangs the modem up.  This ability appears to be
!  implemented on a lot of modems.
!
!  This set of commands is going to be implemented differently on different
!  vendors V.32bis modems.  In this example, Telebit uses S registers.
!  Other modems may use S registers (but different registers), or \
!  commands, or % commands; you get the idea.  (Did I mention that you
!  really, really want to have your modem manual handy?)
!
!  It is important that the modem is configured so that it returns
!  the connected speed, NOT the DTE speed.  The script need to know what the
!  real line speed is in order to set ARA's internal timers.  Some
!  modems don't have the option to display the line speed.  In that case the
!  performance of the connection may not be optimal.
!
! Next, disable MNP and error control
! Turn on internal buffering (for V.32bis), set the delay before disconnect,
! and extended result codes (CTS/RTS flow control was set when we issued &F9,
! so it is not necessary to do it again).
```

```
!
! S180=0 - Turn off all error detection/correction (ARA does MNP and
!          compression itself.  It needs these turned off in the modem).
! S181=1 - Turn *on* DTE <-> line buffering if there is no error control.
!                           Since the modem will be talking to the Mac at
!                           19,200 bps no matter what speed it connects at,
!                           this needs to be on.
! S38=255  - Wait until the modem's buffer is clear OR the other modem
!            disconnects after an ATH is issued before dropping the line.
!            This is done to ensure that all any data in the modem's buffer
!            has been transmitted to the remote modem before it disconnects.
!            If the remote connection does not receive the
!            disconnect packet (usually the last one sent) it could take
!            up to 45 seconds for the remote connection to timeout and
!            disconnect.
! X2 - Issue extended result codes.  This will display busy, connect XXX, etc.
!      X2 will say "CONNECT XXX"  Where XXX is the line speed (as opposed to
!      DTE speed).  This is so ARA can determine what speed the modems are
!      communicating at for timing.
!
@LABEL 5
pause 5
matchstr 1 6 "OK\13\10"
write "ATS180=0S181=1S38=255X2\13"
matchread 30
jump 59
!
!  The modem should now be properly configured.  Now check to see if the user
!  has turned off the modem speaker.  If they have, send an additional command
!  to turn it off.
!
! If speaker on flag is true, jump to label 8.  Otherwise turn off the speaker.
!
@LABEL 6
ifstr 2 8 "1"
pause 5
matchstr 1 8 "OK\13\10"
write "ATM0\13"
matchread 30
jump 59
!
! The modem is ready so enable answering, or originate a call.
!
@LABEL 8
pause 5
ifANSWER 30
note "Dialing ^1" 3
write "ATDT^1\13"
!
!  Be aware that different modems will have different format strings
!  to return connection results.  You need to understand the different possible
!  strings and set this area (and then answer area at label 31) to the
!  appropriate value.  Also, remember that the modem was configured to return
!  the connect speed if possible (The X2 command up at label 5).  It's also
!  useful if the modem can return busy, no dialtone, etc. since the script will
```

```
!  be able to exit quicker and let the user know what is going on.
!
!  Also note that the script waits at the bottom of label 9 for a 70 seconds,
!  rather then looping around.  Why?  Well, if the script re-issues the dial
!  command too soon, that would cause the modem to hang up.  At this point the
!  script should wait a reasonable amount of time for one of these strings to
!  return from the modem and take the appropriate action.
!
@LABEL 9
matchstr 1  11 "CONNECT 1200\13\10"
matchstr 2  12 "CONNECT 2400\13\10"
matchstr 3  13 "CONNECT 4800\13\10"
matchstr 4  19 "CONNECT 7200\13\10"
matchstr 5  14 "CONNECT 9600\13\10"
matchstr 6  20 "CONNECT 12000\13\10"
matchstr 7  18 "CONNECT 14400\13\10"
matchstr 8  50 "NO CARRIER\13\10"
matchstr 9  50 "ERROR\13\10"
matchstr 10 52 "NO DIALTONE\13\10"
matchstr 11 53 "BUSY\13\10"
matchstr 12 54 "NO ANSWER\13\10"
matchread 700
jump 59
!
!  All that is done for different connect speeds is to issue a
!  "CommunicatingAt" command.  Remember, the interface speed is locked
!  to 19,200 bps so the script doesn't want to reset the serial speed after it
!  connects.
!
!  CommunicatingAt tells ARA what the actual line speed is so that it
!  can set it's timers appropriately.  I guess performance would be
!  sub-optimal if this is not set...
!
@LABEL 11
note "Communicating at 1200 bps." 2
CommunicatingAt 1200
jump 15
!
@LABEL 12
note "Communicating at 2400 bps." 2
CommunicatingAt 2400
jump 15
!
@LABEL 13
note "Communicating at 4800 bps." 2
CommunicatingAt 4800
jump 15
!
@LABEL 19
note "Communicating at 7200 bps." 2
CommunicatingAt 7200
jump 15
!
@LABEL 14
note "Communicating at 9600 bps." 2
```

```
CommunicatingAt 9600
jump 15
!
@LABEL 20
note "Communicating at 12000 bps." 2
CommunicatingAt 12000
jump 15
!
@LABEL 18
note "Communicating at 14400 bps." 2
CommunicatingAt 14400
jump 15
!
! Set CTS handshaking ON in the serial port (that's the 1 in the HSReset
! command below )
!
!  The modems have connected, so enable hardware handshaking on the serial
!  port.  If the script is answering a telephone call, just exit right away and
!  starting communicating.  If the script is dialing out, give the other end
!  some time (3 seconds in this example) to get ready to talk to this modem.
!  Exit 0 tells Remote Access that the script was successful in attempting a
!  connection.
!
@LABEL 15
HSReset 0 1 0 0 0 0
ifANSWER 16
pause 30
@LABEL 16
exit 0
!
!  Notice that the @ANSWER label is actually a comment here, and that
!  @ORIGINATE and @ANSWER start at the same place.  What's the point of having
!  separate entry points if they are not used?  Well, in the case of modems,
!  when they dial out or wait for a call, the setup is usually the same.  One
!  reason for separate entry points is when the script is not directly talking
!  to a modem, but maybe to a PBX or terminal server.  It may be necessary to
!  have completely different configuration for answering and originating
!  connections.
!
! @ANSWER
! Set up the modem to answer the telephone.
!
@LABEL 30
write "ATS0=1\13"
matchstr 1 31 "OK\13\10"
matchread 30
jump 59
!
!  What is userhook 1 doing in label 32?  Here's the idea:  Either this script
!  controls a server that is waiting to answer the telephone, or it's waiting
!  for a callback to a connection that was initiated.  AppleTalk Remote Access
!  does a "passive" listen on the serial port (via the Serial Port Arbitrator)
!  so that other communications applications can use the serial port when ARA
!  is not using it.  When a call comes in for a server or callback, there
!  will be about 5-14 seconds while the modems negotiate the connection.
```

```
!  What would happen if a communications application on this Macintosh
!  wanted to use the serial port during that time?  Both connections
!  would fail.  The userhook 1 command tells ARA to mark the serial port in
!  use.  When that happens, applications that want to use the serial port will
!  be told it's busy, and the incoming connection can complete.  With that in
!  mind, the strategy below is:  When the modem receives a ring, jump to label
!  32, issue the userhook 1 command, then jump back up to label 31, wait for
!  the connect result code and continue processing the script.
!
@LABEL 31
matchstr 1  32 "RING\13\10"
matchstr 2  11 "CONNECT 1200\13\10"
matchstr 3  12 "CONNECT 2400\13\10"
matchstr 4  13 "CONNECT 4800\13\10"
matchstr 5  19 "CONNECT 7200\13\10"
matchstr 6  14 "CONNECT 9600\13\10"
matchstr 7  20 "CONNECT 12000\13\10"
matchstr 8  18 "CONNECT 14400\13\10"
matchstr 9  50 "NO CARRIER\13\10"
matchstr 10 50 "ERROR\13\10"
matchstr 11 52 "NO DIALTONE\13\10"
matchstr 12 53 "BUSY\13\10"
matchstr 13 54 "NO ANSWER\13\10"
matchread 700
jump 31
!
@LABEL 32
userhook 1
note "Answering phone..." 2
jump 31
!
!  These are some common error messages when the line is busy, no dialtone,
!  etc. They are documented in the Scripting Language Guide.  When the script
!  exits with a code other than zero, Remote Access knows that the connection
!  failed, and will inform the user with a dialog.
!
! 50: error messages
!
@LABEL 50
exit -6021
!
@LABEL 52
exit -6020
!
@LABEL 53
exit -6022
!
@LABEL 54
exit -6023
!
@LABEL 59
exit -6019
!
! Hang up the modem
! Note:  Why try to enter command mode and hang up the line with ATH, when
```

```
!       de-asserting DTR will always work, and it is used as a last resort
!       anyway?  If DTR is used immediately,  the modem will hang up
!       immediately.  This can have the ill effect of hanging up before all
!       the data in the modem's internal transmit buffer has been sent.
!       It is very desirable to have the last byte of data sent make
!       it out of the modem and across the phone line.  Typically,
!       the last packet sent is the disconnect packet, and if
!       the other side misses this packet, it may have to wait up to 45
!       seconds to hang up.
!
@HANGUP
@LABEL 60
settries 0
HSReset 0 0 0 0 0 0
@LABEL 61
!
!  Here's the basic logic for hanging up: If the modem can be configured
!  to enter command mode when it receives a short break, send a short
!  break.  Send an ATH to hang the line up (and if possible up in the
!  configuration, set the modem to attempt to send all the data in the
!  buffer before it disconnects).  If that fails, it must still be on
!  line, so send the escape sequence to try to drop into command mode.
!  Don't issue a short break again since it did not work the first time.
!  If that fails, de-assert DTR which should force the modem to hang up
!  (make sure the cable is wired properly for this option!).
!  If +++ worked, don't send a short break again; flush the serial port
!  buffer in case the ATH failed due to any stray data hanging around.
!
!  How was this sequence determined?  Trial and error.  Different vendor's
!  modems behave differently when disconnecting.  Some modems will not enter
!  command modem during a disconnect, and the only option is to de-assert DTR
!  to force them to reset.  That's why DTR resets the modem instead of just
!  disconnecting it!  Experiment with this sequence to make it function, but it
!  should work with the majority of the modems available.
!
! Now, since the Telebit modems will drop into command mode when they receive
! a short break (S61=1), issue one here.  This will speed up the disconnect
! sequence by about 5-6 seconds.  Then continue on with normal AT disconnect
! processing.
!
Sbreak
!
! Wait a brief amount of time (1/2 second in this case) so the modem will be
! ready to accept the ATH command.  Pause 1 actually seems to work ok, but
! it's set to 5 just to be safe.
!
pause 5
write "ATH\13"
matchclr
matchstr 1 63 "NO CARRIER\13\10"
matchstr 2 63 "OK\13\10"
matchstr 3 63 "ERROR\13\10"
matchread 30
inctries
iftries 3 63
```

```
! no response, try escape sequence
write "+++"
matchclr
matchstr 1 62 "OK\13\10"
matchread 15
!
! No Response from modem, toggle DTR
!
DTRClear
pause 5
DTRSet
jump 61
!
@LABEL 62
! Pause 1 second to ensure we meet the escape time delay
pause 10
Flush
write "ATH\13"
matchstr 1 63 "OK\13\10"
matchstr 2 63 "NO CARRIER\13\10"
matchstr 3 63 "ERROR\13\10"
matchread 30
jump 61
!
!  Now that the modems have disconnected, and the script has possibly reset the
!  modem, restore the factory settings.  Remember, the script may have hung up
!  the modem in order to get ready for a callback, or it wants to get ready to
!  wait to answer a call again.
!
! recall the factory settings.  Use &F9 again (see note at top of script)
!
@LABEL 63
matchclr
matchstr 1 64 "OK\13\10"
pause 15
write "AT&F9\13"
matchread 30
!
!  Now turn off auto answer if it was turned on to answer a call.  If this
!  script controls a server, the @ANSWER sequence will be called by ARA.
!  One other thing to watch out for here is that some modems expect to
!  talk to the DTE at the last connected speed.  If this is a V.32bis
!  modem and it just finished a connection with a 2400 baud modem, it
!  doesn't necessarily want to talk at 2400 the next time!  Some modems
!  don't exhibit this behavior, so play with it and see what happens.  Finally,
!  since it successfully hung up,  exit the script with a result code of 0 to
!  let Remote Access know everything worked.
!
! Turn off auto answer, set S51 so modem will check interface
! speed on next connection.  If this is not done, the modem
! will not try to autobaud, with the result being it exits the
! script with an error.
!
! S51=254 - Autobaud (19200 bps default)
! S0=0 - Don't try to answer the phone
```

```
!
@LABEL 64
pause 5
matchstr 1 65 "OK\13\10"
write "ATS51=255S0=0\13"
matchread 20
!
@LABEL 65
exit 0
```